VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁVRH A IMPLEMENTACE NÁSTROJE PRO HIERARCHICKOU GRAFICKOU SPECIFIKACI SYSTÉMŮ PRACUJÍCÍCH V REÁLNÉM ČASE

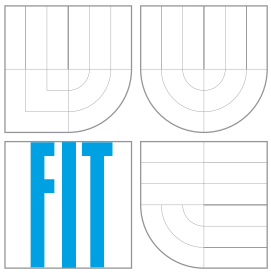
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK GACH

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁVRH A IMPLEMENTACE NÁSTROJE PRO HIERARCHICKOU GRAFICKOU SPECIFIKACI SYSTÉMŮ PRACUJÍCÍCH V REÁLNÉM ČASE

DESIGN AND IMPLEMENTATION OF A TOOL FOR HIERARCHICAL GRAPHICAL SPECIFICATION OF REAL-TIME SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK GACH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOSEF STRNADEL, Ph.D.

BRNO 2009

Zadání práce

1. Seznamte se s principy formální specifikace a verifikace; detailněji pak s metodami předpokládajícími specifikaci v jazyce logiky pracující s reálným časem (RT logiky, RTL).

2. Vytvořte přehled prostředků, metod a nástrojů použitelných při hierarchické grafické specifikaci systémů. Po vzájemné dohodě s Bc. Janem Fiedorem navrhněte blokové schéma a rozhraní nástroje pro hierarchickou grafickou specifikaci RT systémů.

3. Po dohodě s vedoucím vytvořte v přirozeném jazyce neformální slovní specifikaci několika jednoduchých RT systémů.

4. Nástroj (blokově navržený v bodě 2) implementujte a ověřte jeho schopnost generovat RTL specifikaci v podobě zpracovatelné nástrojem vyvíjeným Bc. Janem Fiedorem. Zvažte implementaci metod vedoucích k optimalizaci generované specifikace.

5. Funkčnost nástroje vhodně ověřte, shrňte dosažené výsledky a navrhněte možná rozšíření Vaší práce.

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Josefa Strnadela, Ph.D.

. . . . . .
Marek Gach
25. května 2009

Abstrakt

Práce je zaměřena na návrh a implementaci univerzálního nástroje schopného graficky popsat systémy pracující s reálným časem. Tento nástroj dále umožňuje použít libovolný verifikační přístupu pro ověření výsledného modelu. Je proveden souhrn základních formálních metod popisu systémů založený na matematické logice. Jsou přiblíženy přístupy schopné hierarchicky popsat systémy pracující s reálným časem. Byly navrženy vhodné úlohy pro následné ověření funkčnosti vytvářeného systému.

Klíčová slova

specifikace systému, verifikace, formální metoda, Modechart, UML, real-time systém

Abstract

This thesis is aimed to specify and implement multi-purpose framework able to deal with graphical real-time system specification. This tool allows to use arbitrary verification approach to resulting system model check. Description of basic formal specification methods based on mathematic logic is done. Some well-known hierarchical graphical specifications for real-time systems are depictured. Author proposed suitable cases for functionality examination of resulting framework.

Keywords

system specification, verification, formal method, Modechart, UML, real-time system

Obsah

Kapitola 1

Úvod

Systémy pracující s reálným časem jsou použity pro kontrolu a monitorování v mnoha odvětvích lidské činnosti. Ať už se jedná o letecký či automobilový průmysl, vesmírný výzkum, zdravotnictví nebo jaderné elektrárny. Na systémy tohoto typu jsou kladena nesmírná bezpečnostní omezení. Proto je zde nutno používat metod umožňujících odhalit možná rizika a to nejlépe již v raném stádiu specifikace.

Nevýhodou těchto systémů je fakt, že ve většině případů bývají rozsáhlé a komplexní. Proto definice a použití různých metod pro ověření jejich korektního návrhu a fukcionality je velice obtížná. Také vývojaři takovýchto řešení jsou nuceni u již funkčních realizací provádět občas i velice rozsáhlé změny, které mohou zanést kritické a velice nebezpečné chyby. Proto i z důvodu zkrácení času vývoje a změn nebo snížení nákladů je v těchto případech proces verifikace velice přínosný.

V raných stádiích tohoto odvětví byly využivany jen principy založeny čistě na popisu pomocí matematické logiky. Ale s postupem času se začaly používat vhodnější grafické formalismy. Vznikaly profily založené na UML nebo rozšíření Statechartu v podobě Modechart. Tyto byly plně schopny popsat real-time systémy a zpětně generovat formule v matematické logice.

Tato práce se zabývá vytvořením nástroje, který by vhodně umožnil hierarchicky popsat systémy pracující v reálném čase. Z těchto popisů by vznikly formule vhodně zvolené RT logiky, které by bylo možno následně verifikovat.

1.1 Členění práce

Celá práce je členěna do několika kapitol. První kapitola z důvodu nutnosti pochopení dalšího textu detailně popisuje RTL logiku, která slouží pro popis systémů pracujících s reálným časem na nejnižší úrovni. Je provedena zmínka například i o rozhodnutelnosti a vyjadřovací síle. Následující kapitola přibližuje dva možné formalismy grafického popisu systémů. Jmenovitě se jedná o profily jazyka UML a Modechart. Jsou nastíněny možnosti verifikace. Kapitola s pořadovým číslem 4 ukazuje návrh struktury nástroje, který bude implementován a dále rozsáhle popisuje formalismus Modechart a jeho rozhodovací procedury. U návrhu je důraz kladen na čistotu objektového návrhu. V páté kapitole je důkladně rozebrána implementace nástroje s ohledem na rychlé pochopení nabízených funkcionalit. Šestá kapitola specifikuje 3 modelové úlohy systémů pracujících s reálným časem. Tyto příklady byly úspěšně použity pro ověření funkčnosti nástroje. Poslední kapitola shrnuje dosažené výsledky.

Kapitola 2

Formální specifikace systémů

Tato kapitola je zaměřena na detailní popis matematické logiky RTL. Plné pochopení této logiky je požadováno jejím častým využitím v dalším textu této práce. Jedná se o základní kámen většiny rozšířených systémů umožňujících popis pomocí grafické representace.

2.1 RTL logika

RT logika vznikla jako součást projektu SARTOS v roce 1985. Cílem projektu bylo vytvořit unifikované prostředí umožňující specifikaci a následnou verifikaci správnosti programů pracujících s reálným časem. Následně byl vyvinut i systém algoritmů schopných rozhodnout o splnitelnosti množiny daných RTL formulí. Byla také specifikována formální sémantika technik zjednodušujících analýzu a specifikaci rozsáhlých systémů pracujících právě s reálným časem.

Tento typ logiky primárně vychází z predikátové logiky prvního řádu. Smyslem vzniku této logiky bylo umožnit popsat časové závislosti v systémech pracujících s reálným časem. Čas je zde brán jak relativně tak absolutně. K representaci časových okamžiků dochází celými čísly, tudíž zde dochází k diskretizaci. Tento fakt ale nevadí, jelikož většina výpočetních systémů pracuje s aritmetikou s omezenou přesností výpočtu. Logika je založena na zkoumání pro nás zásadních časových okamžiků v chování systému. Tyto jsou označeny výskytem události. Blíže se na tento princip zaměříme v dalším textu.

2.1.1 Události a akce

Základní dva aspekty na kterých je postavena RTL logika a systémy ji popisované jsou události a akce. Jak již bylo zmíněno dříve, události označují nějaký, pro nás důležitý, časový okamžik v běhu systému pracujícího s reálným časem. Akce jsou poté operace, které jsou popisovány a řízeny událostmi. V porovnání s událostmi akce potřebují pro svůj běh nenulový časový interval a nenulové zdroje cílového systému. Akce je vždy popsána dvěmi událostmi. A to jmenovitě jednou, která vystihuje okamžik zahájení dané akce a druhou, která popíše časový okamžik konce jejího běhu.

Každý analyzovaný systém je definován množinou událostí, které mohou nastat. Tyto události jsou charakterizovány v rámci celého systému jednoznačným jménem. Obecně rozlišujeme tyto tři typy(poslední je ale jen možným zavedeným rozšířením):

- Start a stop události označující zahájení a ukončení akce.
- Přechodové události definující změnu stavové proměnné.

- Externí akce

Start a stop události jednoznačně popisují interval, ve kterém dochází k vykonání určité akce. Položíme-li, že máme akci označenou symbolem A, poté časový okamžik, ve kterém došlo k jejimu zahájení je popsán odpovídající start událostí. Tato událost je pro danou akci A formálně zapsaná jako $\uparrow A$. Ukončení akce je analogicky popsáno událostí $\downarrow A$. Jak je ale patrné, dané relace výskytu se sebou nenesou informaci o čase nebo o tom kolik takovýchto událostí již v systému před tímto časem nastalo.

Přechodová událost popisuje změnu tzv. stavových predikátů, které jsou blíže popsány v následující kapitole. Lze ale nastínit, že tyto predikáty popisují binárně nějakou vlastnost systému. Například můžeme uvažit situaci, že autopilot letadla je zapnut, či nikoliv. Vyjádření je tvaru $(S := T)$ resp. $(S := F)$. Poté přechodové události oznamují změnu hodnoty této vlastnosti systému. Pokud tedy nastane přechodová událost $(S := F)$ znamená to pro nás, že autopilot byl v daném okamžiku vypnut a systém může na tuto situaci adekvátně reagovat například zobrazením informace na řídícím panelu.

Kromě těchto dvou základních tříd událostí lze specifikovat ještě jednu a tou je externí událost. Jedná se o události, které nemohou být způsobeny samotným systémem ale mají dopad na jeho běh. Je voleno speciální označení, které k názvu přidává prefix ve tvaru Ω. Jako ilustraci můžeme uvést událost vzniklou zmačknutím tlačítka BUTTON, která bude ve tvaru $\Omega BUTTON$.

Všechny takovéto události, které mohou být těchto zmíněných tří typů tvoří výslednou množinu všech dostupných událostí námi modelovaného systému.

Predikátový symbol θ vyjadřuje relaci výskytu. Jmenovitě se jedná o relaci na množině $\mathbf{E} \times \mathbf{Z}^+ \times \mathbf{N}$, kde $\mathbf{E}$ je množinou událostí, $\mathbf{Z}^+$ je množinou kladných celých čísel a $\mathbf{N}$ vyjadřuje množinu přirozených čísel. Tento predikátový symbol přiřazuje danému i-tému výskytu události e její čas t. Pro příklad můžeme uvést tvrzení $\theta(\downarrow ACTION,\ 1,\ x)$, které nám říká, že k prvnímu ukončení dané události došlo v čase x.

2.1.2 Stavový predikát

V systémech pracujících s reálným časem je potřeba vhodně modelovat a zjišťovat platnost některých jeho vlastností. A to nejenom jejich hodnotu, která může být například udána jako logická hodnota ale také i čas, ve kterém byla tato hodnota aktuální. Proto RTL logika zavádí pod pojemem stavový predikát formule, které vystihují hodnotu a časový interval, ve kterém dané tvrzení nabývá určité hodnoty.

Předpokládejme, že S je stavový predikát. Ten v daném okamžiku může nabývat logické hodnoty true nebo false. Hodnota tohoto predikátu může být změněna jen pomocí událostí a to jmenovitě události, která mění jeho hodnotu na true, resp. události, která provádí změnu na false. Stavový predikát může být zapsán ve tvaru $S[x,\ y]$. Argumenty x a y, které udávají čas, jsou použity spolu se symboly $[,\ ],\ (,\ )\ \langle$ a $\rangle$, které vyjadřují interval, ve kterém daný predikát nabývá pravdivou hodnotu. Předpokládejme dále, že E_f popisuje přechodovou událost nastavující S na logickou hodnotu true resp. E_f nastavující tuto hodnotu na false. Poté můžeme intervalové symboly popsat takto [4]:

- $[x$ popisuje, že E_t nastala v čase x
- $(x$ popisuje, že E_t nastala před nebo v čase x
- $\langle x$ popisuje, že E_t nastala před časem x
- $y]$ popisuje, že E_f nastala v čase y

- $y)$ popisuje, že E_f nenastala před časem y
- $y\rangle$ popisuje, že E_f nenastala před ani v čase y

Výsledný stavový predikát může být například ve tvaru $S(x,\ y)$ a říká, že v intervalu $\langle x,\ y\rangle$ je predikát pravdivý. Jak je patrné, nezajímá nás jeho hodnota mimo tento interval ať je jakákoliv. Definice nám dále nezakazuje definovat například $S(x,\ x)$ udávající, že predikát je pravdivý v okolí času x nebo také $S\langle x,\ x)$ říkající, že predikát se stane pravdivý a potrvá tak do okamžiku času x.

Každý z takovýchto stavových predikátů lze zapsat pomocí základních operátorů RTL logiky a to například následovně:

- $S[x,\ y] \equiv \exists i\ \theta((S := T),\ i,\ x) \wedge \theta((S := F,\ i,\ y)$
- $S[x,\ y\rangle \equiv \exists i\ \theta((S := T),\ i,\ x) \wedge [\ \forall t\ \theta((S := F,\ i,\ t) \rightarrow\ y < t\ ]$

Ostatní stavové predikáty lze poté definovat obdobně. Můžeme také zavést stavový predikát ve tvaru $\overline{S}$, který výstihuje stavový predikát nabývající v daném intervalu hodnotu false. Na tento zápis lze poté logický aplikovat částečně pozměněné popisy definované výše.

2.1.3 Jazyk RTL logiky

Nyní stručně zavedeme jazyk RT logiky. Výroky jsou representovány formulemi a ty lze vytvořit jen a pouze jen z těchto konstrukcí: pravdivostních symbolů true a false, množiny časových proměnných, množiny proměnných výskytu, množiny konstantních symbolů včetně numerálů celých čísel, množiny konstant události, funkčního symbolu +, predikátových symbolů $<$, $\leq$, $>$, $\geq$, $=$, symbolu relace výskytu, logických spojek $\vee$, $\wedge$, $\neg$ a $\rightarrow$ a univerzálních kvantifikátorů $\exists$ a $\forall$. Term výskytu podle RT logiky je výraz tvořený podle těchto pravidel [4]:

- Konstantní symbol je termem výskytu.
- Proměnná výskytu je termem výskytu.
- Pokud aplikujeme na dva termy výskytu operaci + výsledek je také termem výskytu.

Časový term podle RT logiky je výraz tvořený podle těchto pravidel:

- Konstantní symbol je časovým termem.
- Časová proměnná je časovým termem.
- Pokud aplikujeme na dva časové termy operaci + výsledek je také časový term.

Následně tvrzení v RTL logice je tvořeno takto:

- Pravdivostní symboly true a false jsou tvrzeními.
- Pokud aplikujeme na dva časové termy, resp. termy výskytu operaci rovnosti/nerovnosti, poté vztvoříme tvrzení.
- Pokud i je termem výskytu, t je časovým termem a e je konstanta události, poté $\theta(e,\ i,\ t)$ je tvrzením.

Pomocí tvrzení, logicných spojek a kvantifikátorů jsme poté obvyklým způsobem schopni specifikovat formuli RTL logiky.

2.1.4 Omezené RTL formule

Na základě zkoumámí specifikací systémů pracujících s reálným časem bylo zjištěno, že formule je popisující vykazují jistou podobnost. Na tomto základě byla specifikována speciální omezená třída RT logiky. Bylo zjištěno, že RTL formule jsou většinou representovány nerovnicemi obsahujícími dva termy a číselnou konstantu. Mimo to bylo zjištěno, že tyto formule neobsahují výrazy, které se skládají z funkcí beroucích jako parametr sebe sama. Na základě těchto zjištění můžeme stanovit omezenou RTL formuli ve tvaru:

$$funkce\_vyskytu \ \pm \ konstanta \leq funkce\_vyskytu$$

Což na základě námi již dříve definovaných konstrukcí vytváří formuli tohoto tvaru:

$$@(E_1,\ i) \ \pm \ I \leq @(E_2,\ j)$$

kde E_1 a E_2 patří do množiny událostí, i a j udává pořadové číslo dané události, @ je již dříve zavedenou funkcí výskytu a I je celočíselnou konstantou.

Dále je nutno poznamenat že lze aplikovat tyto dvě následující transformace pro upravení tvaru. Formule tvaru $@(E_1,\ i) \ \pm \ I \ < \ @(E_2,\ j)$ může být zapsaná jako $@(E_1,\ i) \ \pm \ I + 1 \ \leq \ @(E_2,\ j)$. Dále $\neg(@(E_1,\ i) \ \pm \ I \ \leq \ @(E_2,\ j))$ odpovídá $@(E_2,\ j) \ \pm \ I + 1 \ \leq \ @(E_1,\ i)$. Je nutno poznamenat, že konstantu v žádném případě nelze nahradit proměnnou, jelikož poté již není splněn základní námi požadovaný tvar.

2.1.5 Rozhodnutelnost

V této části textu bude zaveden speciální typ aritmetiky. Jmenovitě se jedná o Presburgerovskou aritmetiku, která ve své podstatě splňuje některé pro nás zásadní a zajímavé aspekty. Z důvodu lepšího pochopení si tento systém trochu přiblížíme. Respektive bude přiblížena jeho částečně rozšířená forma. Jmenovitě se jedná o aritmetiku založenou na celých číslech, která postrádá binární operátor násobení a je rozšířena o diferenční operátor nerovnosti. Lze ukázat že tento zmíněný systém, který obsahuje ještě navíc jeden unární nealgebraický predikát je nerozhodnutelný. Lze dokázat, že formule RTL logiky je možné převést do formy vyjádřené touto aritmetikou. Dále je možné pozorovat, že RTL logika je vlastní podmnožinou tohoto systému a tudíž nelze s jistotou zaručit její následnou nerozhodnutelnost. Použitím Downeyova důkazu ale můžeme ukázat, že systém formulí RTL logiky v tomto tvaru je nerozhodnutelný.

Dokázáno ale bylo, že podtřída Presburgerovské aritmetiky, která postrádá neinterpretované funkce je rozhodnutelná a existuje redukce zajišťující eliminaci zmíněných neinterpretovaných funkcí. Uvažujme konjunkci RTL formulí ve tvaru $C_1 \ \wedge \ C_2 \ \wedge \ \ldots \ \wedge \ C_n$, kde C_i ve formě $(Q_i\, L_1 \ \vee \ L_2 \vee \ \ldots \ \vee L_m)$. Pokud uvážíme, že Q_i je zde prefixem skládajícím se z kvantifikátorů a L_j vyjadřuje nerovnost tvaru: $funkce\_vyskytu \ \pm \ konstanta \leq funkce\_vyskytu$, poté pokud prefix Q_i se skládá z obecných kvantifikátorů $\forall$ a $\exists$ můžeme konstatovat, že za těchto předpokladů je daný systém rozhodnutelný.

Tato zjištění jsou velice zásadní, jelikož otevírají novou cestu k možnosti řešení problému splnitelnosti soustavy RTL formulí. Náš problém lze totiž převést do tohoto tvaru a poté se ubírat směrem řešení SMT problému. Zde nám stačí problém specifikovat ve vhodném formátu a předložit libovolnému SMT solveru. Ten rozhodne, jestli je daný systém splnitelný či nikoliv a tím může verifikovat systém s ohledem na libovolné bezpečnostní kritérium (SA).

2.1.6 Vyjadřovací síla

Pomocí lineární temporální logiky nebo RTL jsme schopni popsat přechodové systémy. Tyto systémy jsou většinou vyjádřeny konečným automatem. V RTL lze poté vyjádřit konfiguraci tohoto stavového stroje pomocí událostí a povolené přechody representovat vhodně zvolenými formulemi. Na rozdíl o LTL je real time logika schopna vyjádřit určité aspekty, které v klasické lineární temporální logice popsat nelze. Z tohoto vyplýva, že použití RTL logiky může být v některých případech vhodnější, jelikož její vyjadřovací síla přesahuje možnosti výrokové lineární temporální logiky.

Kapitola 3

Přehled dostupných prostředků

3.1 Specifikace pomocí UML

Při klasickém návrhu systémů nejsme nuceni brát ohled na čas zpracování zprávy nebo čas vykonávání úlohy. Z našeho pohledu tyto události v daném formalismu UML zabirají nulový čas. Ale v systémech pracujících v reálném čase musí být například úlohy vykonávany přesně ve specifikovaném časovém okamžiku (relativním nebo absolutním). Také délka vykonávání je omezena pomocí různých podmínek. Proto v systémech pracujících s reálným časem musíme popisovat například[2]:

- Chování závisle na systémových čítačích nebo hodinách.
- Časové předpoklady kladené na externí prostředí systému. Například čas odpovědi na požadavek či čas potřebný k vykonání určité akce.
- Časově závislé požadavky jako krajní meze vykonávání nebo intervaly mezi dvěmi událostmi.

3.1.1 Funkcionality UML pro popis času

UML definuje metody pro popis času výskytu události nebo časová omezení, která jsou sémantickým aspektem popisujícím absolutní nebo relativní hodnotu času. UML 2.0 vymezuje typy jako časovač nebo hodiny. Zavádí stavové a sekvenční diagramy pro popis časových údajů. Časový diagram poté umožňuje zachytit změny systému v čase.

OCL (Object Constraint Language) je důležitou součástí UML. Umožňuje definovat množinu omezení nad modelem systému. Zavádí invarianty tříd, podmínky pro přechody nebo podmínky, které musí být splněny před nebo po vykonání určité akce. Bohužel OCL ani samotné UML nedovolují definovat časová omezení nad dynamickým modelem.

3.1.2 Dostupné nástroje

Jelikož klasické UML postrádá pokročilé metody pro popis času a časových závislostí a formální základy jsou nedostatečné pro zachycení chování systémů pracujících s reálným čase, vyvstala nutnost definovat jistá rozšíření tohoto formalismu. Z tohoto důvodu vznikly tzv. profily jazyka UML definující formální základy a jistá rozšíření stávajících formalismů této metodologie. V dalším textu budou stručně popsány tři nejpoužívanější profily, které se v jistých aspektech značně odlišují.

SPT profil (UML profile for Schedulability, performance and Time) byl vyžádán a poté přijmut organizací OMG (Object Management Group) jako standard pro modelování systémů pracujících

s reálným časem. Tento krok eskaloval záměr použití technologií založených na objektovém návrhu a UML ve vývoji takovýchto systémů. Profil používá mechanismus stereotype a označené hodnoty (tagged values) pro účel komentování výsledného modelu. Dále obsahuje několik subprofilů pro popis rozličných aspektů jako času, zdrojů či návrhu výkonnosti systému. Například subprofil *RTtimeModeling* definuje model pro popis časových závislostí. Jsou zde specifikovány datové typy Time (čas) a Duration (délka trvání) a mechanismy schopné zachytit jak lokální, tak globální čas. Bohužel díky tomu, že definice daného profilu je velice abstraktní a není plně kompletní, není vhodná pro popis reálných úloh. Dále zde chybí dobře nadefinovaná vazba na funkcionální model, což znemožňuje zpracování specifikace jinými nástroji. Proto je tento profil vázán jen na jeden konkrétní nástroj.

Profil OMEGA-RT upřesňuje jisté aspekty definované dříve zmíněným profilem SPT a vymezuje přesnější fomální sémantiku. Tento přístup je založen na mechanismu událostí, které představují změny stavu systému. Výrazy založené na jazyce OCL jsou použity pro definici časových omezení výskytu jednotlivých událostí. Narozdíl od SPT jsou všechna časová omezení vyjádřena na úrovni diagramu tříd. Jsou definovány metody pro popis času systému, na jejichž základě lze poté pracovat s systémovým časem (např. definovat timeouty, zjišťovat čas systému či měřit čas mezi událostmi). Existují zde pokročilé mechanismy pro popis časových závislostí událostí. Přístup založený na koncepci *Observer* je použit pro zachycení časových omezení mezi více jak dvěmi událostmi. Existuje nástroj umožňující převést tento UML model na časovaný automat, který je poté popsán jazykem IF. Model v takovémto formátu může být následně validován s použitím dostupných nástrojů realizujících model-checking.

Dalším profilem rozšiřujícím UML specifikaci o formální bázi je **TURTLE**. Jedná se o zatím nejpokročilejší přístup ze zde zmíněných. Detailním popisem sémantiky a přístupů k verifikaci tohoto profilu se zabývá další text.

3.1.3 Přístupy k verifikaci

Obecně lze definovat několik možných přístupů k verifikaci systémů zapsaných pomocí grafických profilů derivovaných z UML formalismu. V praxi je například použito několik přístupů současně aby se maximalizoval počet nalezených chyb systému. Přístupy lze zařadit do několika následujících kategorií[2]:

1. Existují přístupy pro převod UML modelů do jazyka Promela. Takto specifikované modely je možné verifikovat například model-checkerem SPIN. Probléme je zde to, že z důvodu použití vlastní definice UML stavového stroje je nemožné verifikovat formule specifikované pomocí lineární temporální logiky.

2. Dalším přístupem je možnost získání axiomatického popisu systému z definované formální specifikace. Poté s použitím theorem proveru může být ověřena pravdivost jakéhokoliv tvrzení o daném systému.

3. Je možné použít klasický přístup simulace, který je velice jednoduchý a flexibilní. Umožňuje odhalení základních chyb v návrhu.

4. UML stavový model může být převeden do podoby časovaných automatů. Následně mohou být použity techniky založené na model-checkingu.

5. Z popisu systému lze získat RTL formule, které je následně možné verifikovat obvyklými technikami.

3.1.4 Nástroj TURTLE

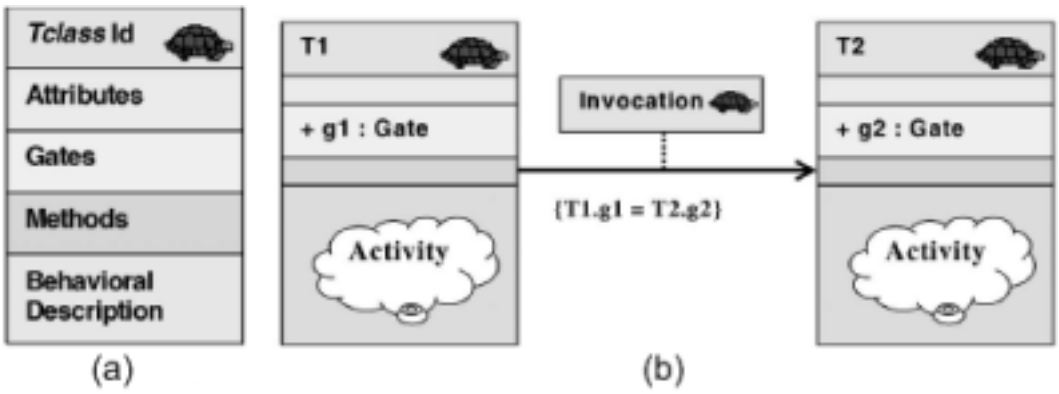
Nástroj TURTLE (Timed UML and RT-LOTOS Environment) vznikl jako prostředek schopný graficky popsat systémy pracujícící s reálným časem a následně umožnit jejich verifikaci. Ikdyž většina v dnešní době používaných popisů je bázovaná na formálních základech, vyvstaly snahy použít některý z již existujících neformálních popisů systémů a dodefinovat potřebná formální rozšíření. Vývoj tohoto nástroje byl založen na neformálním jazyce UML (Unified Modeling Language), který umožňuje různými přístupy popsat systém již ve fázi jeho analýzy a nikoliv za jeho běhu, jak to realizuje většina stávajících formalismů pro verifikaci.

Nástroj TURTLE používá speciální profil jazyka UML, čili základní funkcionality rozšířené nově definovanými. Vytvořený profil může obsahovat elementy základního metamodelu, popisy sémantiky, nově vytvořené notace a pravidla pro překlad modelu, validaci a návrh. Pro vytvoření nových bloků je použit mechanismus stereotype, jakožto způsob definice nových bloků na bázi bloků již existujících, který umožňuje přidání vlastností potřebných pro popis specifického námi řešeného problému.

3.1.5 UML profil programu

V nástroji TURTLE je použito rozšíření dvou základních typů diagramů jazyka UML 1.5. Jedná se o diagram tříd popisující statickou architekturu systému a poté diagram aktivit, který se snaží vystihnout přímo chování jednotlivých komponent.

Rozšířený diagram tříd se skládá z klasických tříd a poté tříd nazvaných Ttřída (Tclass), které vznikly aplikací dříve popsané metody stereotype. V obou případech jsou obsahem tříd definice metod a atributů. Komunikace mezi těmito třídami (manipulace public atributů a volání metod) je limitována na tyto tři kombinace: dvě normální třídy resp. dvě Ttřídy, Ttřídu a normální třídu. Nutné je upozornit, že komunikace mezi dvěmi Ttřídami se uskutečňuje pomocí speciální struktury tzv. brány (gate). Tato konstrukce je atributem dané Ttřídy a umožňuje aplikaci mezitřídních nebo vnitrotřídních synchronizačních mechanismů. Vnitřní chování každé Ttřídy je vyjádřeno diagramem aktivit. Všechny prvky Ttřídy jsou patrné na obrázku 3.1(a).



Obrázek 3.1: Struktura Ttřídy(a) Synchronizace operátorem Invocation(b) [1]

Profil programu rozšiřuje klasický diagram aktivit jazyka UML o synchronizační a temporální operátory. A umožňuje tak plně vystihnout chování systému pracujícího v reálném čase.

Synchronizační operátory napomáhají k synchronizaci aktivit jak v rámci jedné Ttřídy tak i mezi dvěmi třídami tohoto typu. V prvním případě hovoříme o interní a v druhém případě ex-

terní synchronizaci. Jak již bylo zmíněno, tak v tomto případě jsou všechny synchronizační aktivity prováděny pomocí bran dané třídy. Při volání přes bránu dané třídy mohou nastat tyto tři případy:

- Brána není synchronizována s žádnou jinou. V tomto případě je modelována interní akce.
- Dochází k interní synchronizaci pomocí dané brány. K této situaci může docházek při paralelním běhu více aktivit v rámci jedné třídy. K synchronizaci poté dochází například v okamžiku, kdy jsou obě aktivity připraveny provést jistou akci.
- Brána je externě synchronizována. K této situaci dochází při synchronizaci mezi dvěmi třídami. Kde je definováno pomocí operátoru Synchro spojení mezi dvěmi Ttřídami a je specifikováno, které brány se mají mezi sebou synchronizovat.

V rámci vzájemné synchronizace může docházet k předávání dat. Například jedna třída bude data posílat a druhá realizuje jejich příjem. V tomto je nutno podotknout, že jsou rozlišeny dva typy bran, kterými jsou brány vstupní a výstupní. Rozdělení je dáno tím, jestli umožňují data přijímat nebo zasílat. Na typ zpráv není brán zřetel a neexistují například žádné seznamy povolených typů zpráv pro dané brány, jak je tomu u většiny dalších formalismů. Z toho vyplývá, že pomocí bran lze v danou chvíli posilat jakýkoliv typ zprávy.

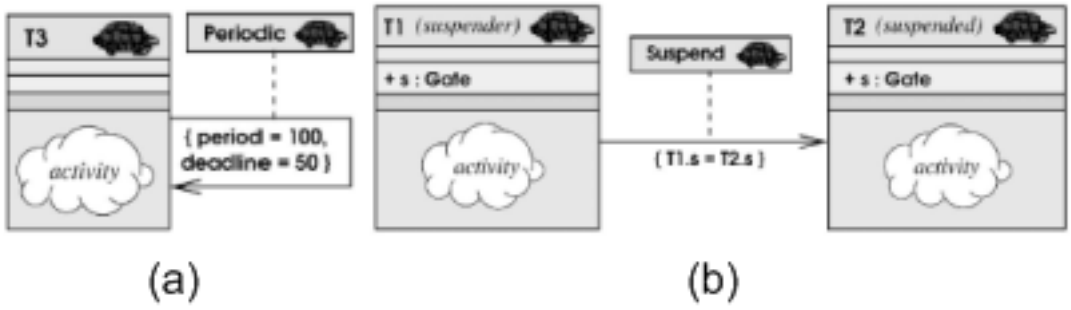
Z důvodu nutnosti modelování časových omezení systému byly do diagramu aktivit zavedeny temporální operátory. V daném profilu se jedná celkem o čtyři jejich základní typy. Deterministické zpoždění je charakterizováno konstantou d, která udává v časových jednotkách opoždění vykonávání dané úlohy. Dálším typem temporálního operátoru, který již nativně není definován v jazyce UML, je nedeterministické zpoždění. Jeho parametrem je konstanta t, která udává maximální hodnotu opoždění vykonávání dané úlohy. Poté reálné opoždění je dáno nějakou náhodnou hodnotou v intervalu $(0, t >$. Ikdyž nativně profil programu TURTLE nedefinuje intervalové zpoždění, čili zpoždění dané nějakým časovým intervalem, lze ho realizovat kompozicí dvou předešlých operátorů. Pokud je nutno například danou akci opozdit minimálně o t_1 a maximálně o t_2 časových jednotek, je toho možno docílit serializací deterministického zpoždění s parametrem t_1 a nedeterministického zpoždění s parametrem $t_2 - t_1$. Dalším temporálním operátorem je časově omezená nábídka (time-limited offer). Udává možnost uskutečnění nějaké předem definované akce před uplynutím jistého časového kvanta. A nakonec posledním operátorem, který nabízí nástroj TURTLE je operátor zachycení času (time capture operator). Tento operátor umožňuje zachytit množství času od momentu, kdy Třída nabízí možnost vykonání akce(např. akce je připravena v vykonání ale nemůže být vykonaná s důvodu synchronizace s jinou akcí, která nemůže byt v danou chvíli uskutečněna) a časem samotného vykonání dané akce. S použitím těchto temporálních operátorů je možné modelovat pokročilé temporální chování jako například timeouty, watchdogy a ostatní mechanismy používané v systémech pracujících v reálném čase.

3.1.6 Rozšířený UML profil programu

Nástroj TURTLE definuje pokročilejší rozšířený profil UML. Ten definuje další zajímavé rozšíření stávajícího profilu. V základním profilu zcela chybí jakýkoliv operátor vystihující volání metod. Z toho důvodu je definován operátor Invocation. Základní profil rozšiřuje diagram aktivit o velice důležité temporální operátory ale stále chybí RT operátory na úrovní diagramu tříd. Proto byly dodefinovány kompozitní operátory Periodic a Suspend/Resume.

Volání metod je základním rysem objektově orientovaných jazyků a UML diagramu tříd zvláště. S použitím základního profilu bylo nutno volání metod modelovat pomocí dvou Synchro operátorů. A to ještě zde nastává problém, že je nutno kontrolovat korektní výměnu dat. Z důvodu zjednodušení

byl zaveden operátor Invocation, který umožňuje jisté Ttřídě vložit do svého toku řízení aktivity jiné třídy. Proto se tento operátor v daném případě různí od operátoru Synchro, který prováděl složitou synchronizaci mezi dvěmi toky řízení. Ukázka operátoru Invocation je patrná na obrázku 3.1(b). První třída je zde volající a druhá volaná. Je také patrná formule v jazyce OCL specifikující, které brány budou použity.



Obrázek 3.2: Operátor Periodic(a) Operátor Suspend(b) [1]

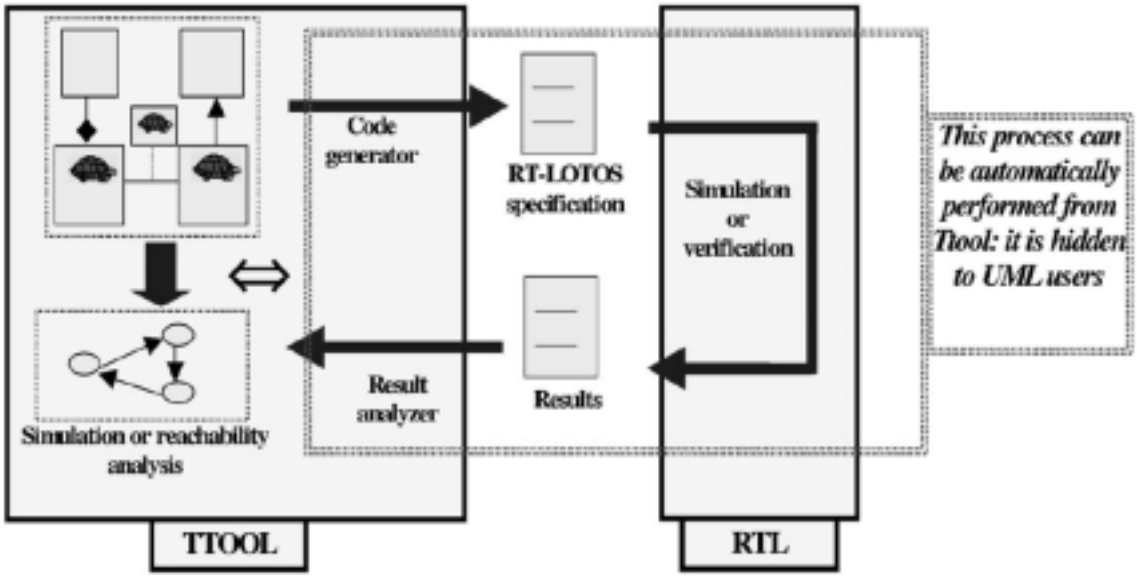
Základní profil postrádá způsob popisu periodických akcí. Je definován operátor Periodic, který umožňuje popsat periodické chování akcí Ttříd. Tento asociace může být definován jak mezi dvěmi Ttřídami, tak i reflexivně v rámci jedné Ttřídy. K jejímu popisu slouží parametr vyjadřující délku periody a deadline délky exekuce. V prvním případě pokud tato asociace existuje mezi dvěmi Ttřídami, poté po výkonání běhu výchozí třídy, je akce cílové třídy vykonána periodicky s danými parametry. K ukončení periodického běhu může vést například aplikace asociace typu Synchro. Ttřída je schopna aplikovat asociaci typu Periodic směrovanou sama na sebe. V tom případě prakticky dojde k vytvoření smyčky běhu (3.2(a)).

Jsou dodefinovány dva operátory Suspend a Resume. Kompozitní operátor Suspend umožňuje pozastavení běhu Ttřídy. Pokud je tímto způsobem aplikován tento operátor, tak ikdyž běh dané Ttřídy je pozastaven, její čítač času běží nadále. Poté může exekuce takováto Ttřídy pomocí operátoru Resume pokračovat dále a to od stavu v jakém se nacházela při pozastavení. Příklad operátoru Suspend je na obrázku 3.2(b). Spolu s těmito dvěmi operátory je pro již dříve zmiňované temporální operátory definováno chování schopné při čekání pozastavit běh dané Ttřídy a to na úrovní diagramu aktivit. Jmenovitě je definováno deterministické pozastavitelné zpoždění, nedeterministické pozastavitelné zpoždění atd.

3.1.7 Funkcionality nástroje

Nástroj TURTLE nabízí editor diagramu tříd, diagramu aktivit, kontrolu syntaxe, generátor kódu ve formátu RT-LOTOS a grafický analyzér verifikace a simulace. Všechny tyto funkcionality jsou součástí modulu TTool. Tento modul je bezprostředně spojen se součástí RT-LOTOS, která umožňuje zpracovat specifikaci generovanou nástrojem TTool. Poté je schopna provést jak verifikaci na základě úplné analýzy, tak stochastickou simulaci. Výsledky těchto kroků jsou zasílány zpět a jsou vhodně zpracovány. Zde probírané schéma nástroje TURTLE je zobrazeno na obrázku 3.3 níže.

Převod modelů do jazyka RT-LOTOS je značně náročný, jelikož pro většinu operátorů a konstrukcí neexistuje v tomto jazyce vhodný protějšek a je potřeba definovat postupy, které umožní tuto situaci vhodně vyřešit. Následná verifikace na základě úplné analýzy je paměťově náročná a bere v potaz například i syntakticky nesprávné modely. Vyskytuje se zde negativní efekt stavové expoloze, která násobí paměťovou náročnost. Proto je tento typ analýzy aplikován většinou jen na



Obrázek 3.3: Struktura nástroje TURTLE [1]

klíčové části a na algoritmy důležité pro běh systému. Doplňkovou komponentou verifikace je simulace. Ta umožňuje částečně prozkoumat stavový prostor úlohy. Přispívá k analýze komplexních rozsáhlých systémů, kde nelze z důvodu nedostatku paměti provést nad celým systémem verifikaci. Dalším možným přístupem, který je zde dostupný je analýza grafu dostupnosti. Ten je sestrojen na základě detekci přechodů systému a synchronizací. Poté je možné provést formální důkaz s použitím grafu dostupnosti a námi zkoumané logické formule popisující zkoumanou podmínku systému. K této analýze je použit model checker Kronos, který z takto poskytnutých hodnot je schopen vyhodnotit jestli daná formule je pravdivá pro graf (resp. systém) či nikoliv.

3.2 Modechart

Ikdyž RTL logika je velice použitelným způsobem popisu systému, tak díky tomu, že vychází jen ze slovní specifikace je velice náchylná na chyby a nedostatky v návrhu. Také následná úprava některých vlastností systému popsaného pomocí RTL formulí je značně náročná. Sémantika modechartu je založena na RTL a umožňuje převod modelu do formulí RTL logiky. Modechart má hierarchické uspořádání a výsledné RTL formule tyto vlastnosti zachovávají, což je jedna z velkých předností tohoto formalismu. Dále je výhodou to, že je zde možné aplikovat do značné míry abstrakci, která umožňuje popis a následnou verifikaci jednotlivých subsystémů rozsáhlých modelů.

3.2.1 Dostupné nástroje

Jazym pro grafickou hierarchickou specifikaci vznikl stejně jako RTL jako součást projektu SARTOR. Všechny další výzkumy se shromažďují jen kolem tohoto projektu. Byly definovány přístupy schopné definovat z modelu popsaného modechartem odpovídající formule jazyka RTL. Samotný program pro tvorbu grafické specifikace se skládá ze dvou částí. První umožňuje vytváření, zobrazování a modifikace grafické specifikace systému. Druhá části je databázový správce, ze kterého jsou pomocí zasílání zpráv získávána potřebná data první komponentou.

Je použit nástroj TRANS pro generování RTL formulí z modechart specifikace. Tento nástroj převádí Modechart do textové ASCII specifikace na jejimž základě jsou poté deklarativně pomocí dotazů generovaný RTL formule. Vstupní textová specifikace umožňuje případně použití i jiného nástroje pro specifikaci modechart než stávajícího.

Nástroj SARTOR byl použit pro verifikaci rozličných systémů pracujících s reálným časem například v odvětví elektroniky, vesmírného výzkumu nebo letectví.

3.2.2 Přístupy k verifikaci

Specifikaci generovanou pomocí formalismu modechart je možno verifikovat rozličnými způsoby. Obecně lze definovat tyto základní:

1. Exitují přístupy umožňující realizovat simulaci systému. Takovýto přístup je schopen odhalit jednoduché chyby již v raném stádiu vývoje.

2. Je možné získat popis systému pomocí RTL formulí a následně provést klasické kroky verifikace vůči bezpečnostnímu tvrzení.

3. Často používaným přístupem je použití model-checkingu nad grafem representujícím běh systému.

4. Je možno použít tzv. inkrementální debuging. Což je přístup založený na matematické logice, který ověřuje pravdivost bezpečnostního tvrzení na základě postupného přidávání omezení. Výhodou je, že splnitelnost modelu není přepočítavaná vždy pro model jako celek.

5. Na graf běhu systému je možno použít theorem prover. A na základě matematické logiky ověřit pravdivost logické formule vůči specifikaci systému.

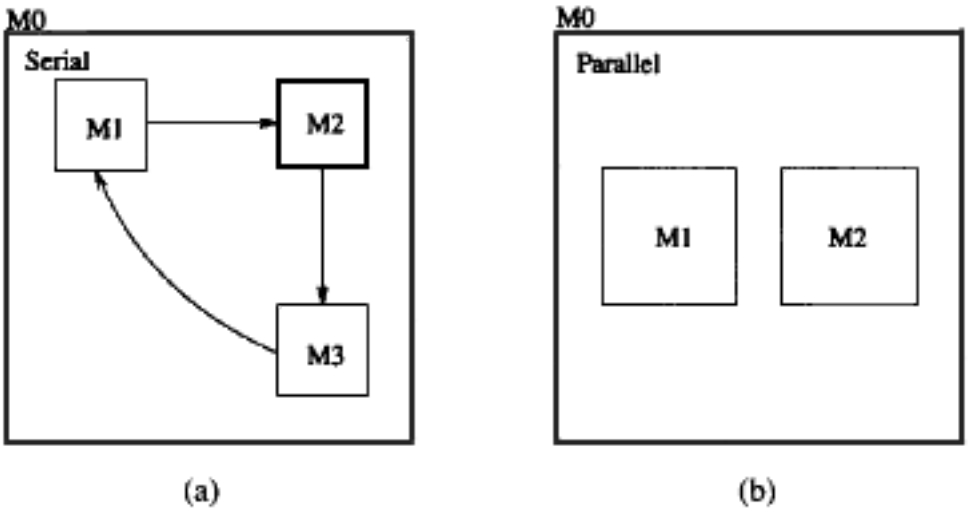
3.2.3 Definice formalismu

Modechart je grafickým jazykem schopným popsat systém pracující s reálným časem. Hlavními prvky popisu jsou mody (značené jako čtverec) a přechody (representované hranou mezi mody). Skupina modů představuje (kontrolní) stav systému a přechody popisují tok řízení.

Mody jsou implicitně značeny čtvercem a v rámci systému jsou rozpoznatelné pomocí svého jedinečného názvu. Dalším parametrem je příznak aktivity modu. Ten je aktivní od doby vstupu do něj a aktivita trvá těsně do okamžiku opuštění. Mode je aktivní i neaktivní v okamžiku vstupu do něj nebo jeho opuštění. Obecně jsou definovány tři typy modů: atomické, sériové a paralelní. Pro ilustraci nám v dalším textu poslouží obrázek 3.4.

Atomické mody jsou základními stavebními prvky modechartu. Tyto bloky postrádají vnitřní strukturu a representují primitivní součásti systému. Pokud si představíme modechart jako stromovou strukturu, tak tyto bloky budou tvořit listy. Na obrázku 3.4 jsou těmito prvky např. M_1, M_2 či M_3.

Sériový mode je speciálním typem složeného modu. Skládá se z jednoho nebo více synovských prvků spojených do série. Každý složený mode obsahuje atribut říkající, jestli se jedná o sériový nebo paralelní typ. Na obrázku 3.4(a) je M_0 kompozitním sériovým modem. Mody M_1, M_2 a M_3 jsou v daném případě atomické komponenty spojeny do série. Mimoto M_2 je označen tlustější čarou, což znamená, že je výchozím modem. Takže pokud je veden přechod do M_0, je realizován vstup do M_0 a zároveň do M_2. Nic nebrání ale tomu, aby byl veden externí přechod například do M_1 či M_3 a v této situaci nemusí být systém iniciálně ve výchozím modu M_2. Z tohoto je patrné, že pokud je systém v modu M_0 poté je současně v M_1, M_2 či M_3. Ikdyž je na obrázku patrné, že



Obrázek 3.4: Paralelní a sériový mode. [5]

všechny přechody vedou jen v rámci jedné úrovně M_0, specifikace modechart nezabraňuje vedení přechodů mezi různými úrovněmi.

Paralelní mode obsahuje 0(atomický mode) nebo více synovských komponent. Paralelní kompozice indikuje, že systém se nachází ve všech synovských modech souběžně. Na obrázku 3.4(b) vidíme příklad kompozitního paralelního modu M_0, který obsahuje paralelizované mody M_1 a M_2. Přechody mezi mody v paralelní kompozitní komponentě nejsou ve specifikaci modechart povoleny. Při vstupu do paralelního modu jsou aktivovány všechny jeho synovské komponenty. Z toho vyplývá, že se zde nespecifikuje v porovnání se sériovým případem výchozí stav. Uvážíme-li, že v příkladě na obrázku je systém v modu M_0, poté je zároveň v M_1 i M_2. Jakýkoliv přechod realizující opuštění modu M_0 musí opustit i všechny jeho synovské komponenty.

Přechody v modechartu směřující z jednoho modu do druheho ovlivňují tok řízení systému. V modechartu je přechod zakreslen orientovanou hranou a popisuje tok řízení ze zdrojového modu do cílového. V rámci jedné úrovně podle definice mohou existovat přechody jen v sériových kompozitních modech. Pro přechody mezi různými úrovněmi žádné takovéto omezení neplatí. Přechod je děj, který zabírá nulový časový okamžik. Čili z definice události, která musí trvat nulový čas, vyplývá, že přechod je událostí. Narozdíl od tohoto, informace o aktivitě modu událostí není, jelikož trvá nenulová čas. Formálně je přechod popsán jako $M_s \rightarrow M_d$, kde M_s je zdrojový a M_d cílový mode. Vstup do modu M je popsán jako $\rightarrow M$. Opuštění modu je poté značeno jako $M \rightarrow$. Oba tyto děje zabírají nulový čas systému. Každý přechod obsahuje podmínku, po jejimž splnění je proveden. Podmínka je popsaná v disjunktní normální formě ve tvaru: $c_1 \vee ... \vee c_k$. Pro každý disjunktní literál platí, že je buď spouštěcí podmínkou nebo časovou podmínkou.

Spouštěcí podmínka je vyjádřena v konjunktivní normální formě ve tvaru $e_1 \wedge ... \wedge e_n$ a každy prvek e je událost nebo predikát. Aby byla spouštěcí podmínka splněna je nutno aby v danou chvíli nastaly všechny nutné události a zaráz byly splněny i všechny potřebné predikáty. Formálně lze do konjunkce dosadit [3]:

1. Událost $\rightarrow M$ je splněna po vstupu do modu M.
2. Událost $M \rightarrow$ je splněna po opuštění modu M.
3. Událost $M_1 \rightarrow M_2$ je splněna pokud nastane přechod $M_1 \rightarrow M_2$.

4. Predikát $M == true$ je splněn pokud mode M je aktivní.

5. Predikát $M == false$ je splněn pokud mode M není aktivní.

6. Seznam predikátů $\{(M_1, ..., M_n)\}$ je splněn pokud jakýkoliv mode ze seznamu je v daný okamžik aktivní.

7. Seznam predikátů $\{< M_1, ..., M_n)\}$ je splněn pokud jakýkoliv mode ze seznamu je aktivní a byl aktivní minimálně jednu časovou jednotku.

Časovaná podmínka je specifikována zpožděním a krajní mezí (deadline). Tato podmínka poté vypadá následovně: $(r,\ d)$, kde $r \leq d$ a obě hodnoty jsou kladnými celými čísly. Zavedeny jsou zkrácené notace, kde $(delay\ r)$ znamená $(r,\ \text{inf})$, $(deadline\ d)$ odpovídá $(0,\ d)$ a $(alarm\ r)$ vystihuje $(r,\ r)$. Pokud máme časovou podmínku $(2,\ 5)$, tak udává, že přechod může nastat 2 časové jednotky po vstupu do modu a ne později jak 5 časových jednotek po této události.

3.2.4 Získání RTL formulí

Modechart byl vyvinut hlavně z důvodu ulehčení popisu systému, kde reálně byla primárně použita jen temporální logika. Jelikož tento přístup vycházel z RTL logiky, je zajisté jasné, že existují postupy pro získání RTL formulí z popisu pomocí modechart. Výhodou takto vzniklých formulí je fakt, že zachovávají hierarchickou strukturu systému a popisují všechna jeho omezení vyplývající i z podstaty definice formalismu modechart. Zmíněná hierarchická struktura umožňuje do modelu či následných formulí RTL zavést jistou úroveň abstrakce, která umožní například vypustit nepodstatné subkomponenty systému. Pro získání RTL formulí je nutno použít postup definován v několika bodech níže [7]:

1. Pro každý(jedinečný) spouštěcí přechod (triggering transition) tvaru $M \rightarrow N$ se spouštěcí podmínkou C přidáme formuli:

$$\forall t\ \ M(t,\ t)\ \wedge\ C\ \Rightarrow\ \exists j\ \ @(M \rightarrow N,\ j)\ =\ t$$

2. Pro každý(jedinečný) časovaný přechod (timing transition) tvaru $M \rightarrow N$ s časovanou podmínkou $(r,\ d)$ definujeme formuli:

$$\forall t\ \ M[t,\ t)\ \Rightarrow\ \exists t',\ \exists j\ \ @(M \rightarrow N,\ j)\ =\ t'\ \wedge\ B, \text{kde } B\ =\ (t + r \leq t')\ \wedge\ (t' \leq t + d).$$

3. Každé dva přechody explicitně opouštějící mode musí splňovat vzájemné vyloučení(mutual exclusion). Uvažme dvě přechodové proměnné $e_1,\ e_2$ omezení které popisuje vzájemné vyloučení je ve tvaru:

$$\forall i,\ \forall j\ \ @(e_1,\ i)\ \neq\ @(e_2,\ j)$$

4. Pokud M je seriový mode (a $M_i \in children(M)$) a systém opouští M v čase t poté je přidána následující podmínka:

$$\forall t\ \ M(t,\ t]\ \Rightarrow\ \bigwedge_{i=1}^{n}(M_i(t,\ t))\ \Rightarrow\ M_i(t,\ t])$$

5. Pokud M je paralelní mode (a $M_i \in children(M)$) a systém opouští M v čase t poté je přidána následující podmínka:

$$\forall t \;\; M(t,\, t] \;\Rightarrow\; \bigwedge_{i=1}^{n} M_i(t,\, t]$$

6. Přechodová podmínka pro vnořené mody na úrovni M (kde M je seriový mode) pro každý mode $M' \in children(M)$ jsou definovány následující formule:

   - $\forall t \;\; M'(t,\, t) \;\Rightarrow\; \bigwedge_{i=1}^{n}(C_i \rightarrow (T_i \;\vee\; M'(t,\, t]))$
   - $\forall t \;\; M'[t,\, t) \Rightarrow \bigwedge_{i=1}^{n}[(T_{n+i} \;\wedge\; B_i) \;\vee\; (\exists t' \;\; M'[t,\, t'] \;\wedge\; t' \leq t + d_i)]$
   - $\bigwedge_{i=1}^{n+m} T_i \rightarrow R_i$
   - $\forall t \;\; M'(t,\, t) \rightarrow \{formule\ urovne\ M'\}$

   Kde T_i je predikát popisující výskyt přechodové události spjaté s i-tým přechodem vedoucího z M' a každé R_i je konjunkcí:

   - predikátů popisujících explicitní opuštění pro i-tý přechod a
   - predikátů vystihujících explicitní a implicitní vstupy do modů pomocí i-tého přechodu.

7. Přechodová podmínka pro vnořené mody na úrovni M (kde M je paralelní mode) a $M' \in children(M)$ jsou definovány následující formule:

   - $\forall t \;\; \bigwedge_{i=1}^{n}(C_i \Rightarrow (T_i \;\vee\; M(t,\, t]))$
   - $\forall t \;\; M[t,\, t) \Rightarrow \bigwedge_{i=1}^{m}[(T_{n+1} \;\wedge\; B_i) \;\vee\; (\exists t' \;\; M[t,\, t'] \;\wedge\; t' \leq t + d_i)]$
   - $\bigwedge_{i=1}^{n+m} T_i \Rightarrow R_i$
   - $\forall t \;\; M'(t,\, t) \Rightarrow \{formule\ urovne\ M'\}$

První dvě podmínky vystihuji klasické dva druhy přechodů. Čili ty založené na splnění jisté podmínky a poté ty, jež jsou popsány časovými omezeními. Třetí podmínka zavádí vzájemné vyloučení, čimž umožňuje v jedné chvíli vždy volit jen jeden přechod opouštějící mode. Další dvě podmínky zaručí opuštění všech synovských modů v případě že přechod vede ven z rodičovského modu (paralelního či sériového). Poslední dvě podmínky popisují přechody vycházející ze synovských modů rodiče M. Jedná se jak o přechody se spouštěcí podmínkou, tak časově omezené přechody. Další formule popisuje opuštění a vstupy mezi mody v různých úrovních.

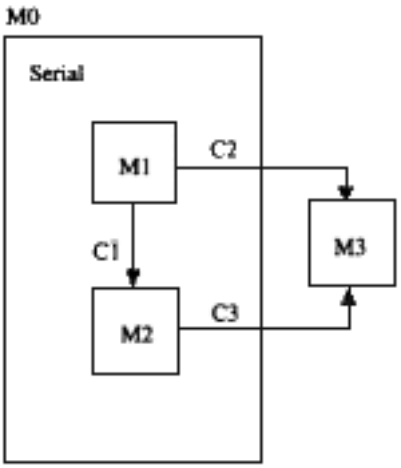
Položmě, že S je modechartem. Poté RTL(S) je množinou omezení sdruženým s S dodržujícím sémantiku výše. Pokud máme sekvenci vstupních událostí $I = i_1,\; i_2, ...$ a posloupnost výstupních událostí $O = o_1,\; o_2,\; ...$, které jsou výpočtem S jen a právě tehdy když formule:

$$\forall t \;\; @(i_t) \;\wedge\; @(o_t,\, t) \;\wedge\; RTL(S) \text{ je splnitelná,}$$

a kde $@(i_t,\, t)$ a $@(o_t,\, t)$ popisují, že vstupní a výstupní události na pozici se vyskytují přesně v čase t.

3.2.5 Příklad získání RTL formulí z modechartu

Z důvodu demonstrace postupu uvažme jednoduchý příklad na obrázku 3.5. Pomocí výše zmíněných pravidel nagenerujeme odpovídající popis v logice RTL pro daný grafický popis v jazyce modechart. Jelikož podmínky generované na základě bodu 1, 3 a 4 jsou zřejmé, zaměříme se na nejsložitější část a to v danem případě formule generované postupem vystihnutým v bodě 6. Podle daného postupu nadefinujeme formule jak pro M_1 tak M_2, jelikož podle definice jsou syny modu M0. Výsledné formule mají tvar:



Obrázek 3.5: Jednoduchý příklad Modechart [5]

- $M_1(t,\ t)\ \wedge\ C_1 \Rightarrow \exists i\ @((M_1 \rightarrow M_2),\ i) = t\ \vee\ M_1(t,\ t]$
- $M_1(t,\ t)\ \wedge\ C_2 \Rightarrow \exists i\ @((M_1 \rightarrow M_3),\ i) = t\ \vee\ M_1(t,\ t]$
- $\forall i\ @((M_1 \rightarrow M_3),\ i) = t \Rightarrow M_1(t,\ t]\ \wedge\ M_0(t,\ t]\ \wedge\ M_3[t,\ t)$
- $\forall i\ @((M_1 \rightarrow M_2),\ i) = t \Rightarrow M_1(t,\ t]\ \wedge\ M_2[t,\ t)$
- $M_1(t,\ t) \Rightarrow \{formule\ urovne\ M1\}$
- $M_2(t,\ t)\ \wedge\ C_3 \Rightarrow \exists i\ @((M_2 \rightarrow M_3),\ i) = t\ \vee\ M_2(t,\ t]$
- $\forall i\ @((M_2 \rightarrow M_3),\ i) = t \Rightarrow M_2(t,\ t]\ \wedge\ M_0(t,\ t]\ \wedge\ M_3[t,\ t)$
- $M_2(t,\ t) \Rightarrow \{formule\ urovne\ M2\}$

Jak bylo viditelné na předešlém příkladu, tak z modechartu lze bez značných problémů získat použitelné RTL formule. Je ale nutno poznamenat, že například s rostoucím počtem modů nebo velikostí jejich zanoření bude značně narůstat počet formulí. To ale zajisté nebrání v dalším použití těchto formuli v následné verifikaci. Existují matematické postupy pro převod takovýchto formulí do Pressburgerovské aritmetiky, jmenovitě do formy omezených RTL formulí. Nad těmito již lze obyklými způsoby provést verifikační proces. A to buď klasickou analýzou CG grafu nebo s použitím matematické logiky, čili zjištěním splnitelnosti generované teorie pomocí SMT solveru.

Kapitola 4

Analýza

4.1 Přístup k verifikaci modechart

Jak již bylo zmíněno, pro účel grafického popisu systému pracujících s reálným časem byl pro vyvíjený nástroj zvolen formalismus Modechart. Již v jedné z předešlých kapitol byly přiblíženy základní aspekty a vlastnosti tohoto přístupu specifikace. Byl definován základní koncept a přístup, jakým lze danou grafickou specifikace převést do podoby RTL formulí. Nad generovanými formulemy RTL logiky lze samozřejmě provádět základní ověřování funkčnosti systému, ale pro účel vyvíjeného programu je tento přístup relativně složitý a není nad ním vybudován potřebný aparát pro zjišťování a analýzu pokročilejších vlastností systému. Proto bylo nutno přikročit k odlišnému přístupu řešení problému analýzy a verifikace systémů specifikovaných Modechartem. Z tohoto důvodu bude přiblížen formální model formalismu Modechart. Nad tímto poté bude popsán aparát umožňující analýzu běhu systému.

4.1.1 Formalismus Modechart

Z důvodu pochopení dalších definic je nutno nastínit formální definici formalismu Modechart pro hierarchický popis systémů pracujících s reálným časem. Celkem existují dva formální pojetí, které sdílejí většinu společných rysů. Specifikace popsaná v [9] sice velice dobře nastiňuje tuto problematiku, ale v dalším pokračování bude stavěno na formálním aparátu specifikovaném v [7]. Tento přístup formální specifikace vychází z prvně zmíněného pojetí a některé aspekty definuje trochu odlišným a přijatelnějším způsobem.

Modechart je pěticí tvaru $\langle\ \mathcal{M},\ \{\ S, P, initial\},\ \sqsubseteq,\ \mathcal{T},\ \varepsilon\ \rangle$ se složkami definovanými následovně:

- $\mathcal{M}$ je (konečnou) množinou modu.
- $S \subset \mathcal{M}$ je množinou seriových modu a poté $P \subset \mathcal{M}$ je množinou paralelních modu a dále musí platit $S \cap P = \emptyset$ a $S \cup P \neq \mathcal{M}$. Totální funkce $initial : S \rightarrow \mathcal{M}$ označuje mode (počáteční nebo výchozí) odpovídající seriovému modu.
- $\sqsubseteq$ je relací částečného uspořádání v $\mathcal{M}$ definující strom napříč elementy množiny $\mathcal{M}$. Dále je použita notace $\sqsubseteq$ a $\not\sqsubseteq$. Zápis $m \sqsubseteq n$ vystihuje, že m je obsažen v n neboli m je potomek n či n je předek m. Je potřeba aby $S \cup P = \{m \in \mathcal{M} \mid n \sqsubseteq m, pro\ n \in (M)\}$.
- $\mathcal{T} = (M) \times (M) \times MTE$ je množinou přechodů. Pro přechod (s,t,e), s je zdrojovým modem, t je cílovým modem přechodu a e je přechodovým výrazem Modechartu (MTE).
- ε je (konečnou) vstupní množinou externích vstupních signálů.

Množina externích výstupních signálů modechart specifikace je definována jako označení a je dána následujícím sjednocením čtyř množin označení: $\mathcal{L} = M^+ \cup M^- \cup M^\perp \cup M^\rightarrow$, kde množina označení vstupu do modu M^+ pro $M \subseteq (M)$ je definována jako: $M^+\{m^+ \mid m \in M\}$ a pot0 m^+ je označením vstupu do modu m. Následně množina označení M^- a $M^\perp$ je definována podobně a to pro označení opuštění resp. aktivity modu. Z hlediska těchto označení je pro každý mod m zachycena evoluce systému a to aktivita, vstup a opuštění. Dodatečně, každý přechod systému má označení ve tvaru: $\mathcal{M}^\rightarrow = \{s \rightarrow t \mid (s,t,e) \in \mathcal{T}, pro\ e\ v\ MTE\}$

Přechodový výraz, na základě kterého je rozhodováno o povolení přechodu je možno definovat pomocí BNF gramatiky, kde MTE je počáteční symbol:

$$\begin{aligned} MTE &::= TimingC | TrigC \\ TimingC &::= (lb, ub) \\ TrigC &::= Event \wedge TrigC \mid Event \\ Event &::= e \mid \overline{e} \mid m^+ \mid m^- \mid m \rightarrow n \\ MSet &::= m^\perp, MSet \mid m^\perp \end{aligned}$$

V této gramatice vystupuje horní a spodní časové omezení přechodu, kde $lb, ub \in \mathbb{N}$, e popisuje v výskyt externí události a poté $\overline{e}$ její absenci. Mody specifikace rozumíme $m, n \in \mathcal{M}$ a výraz $m \rightarrow n$ popisuje vedení přechodu z m do n. Výraz tvaru (lb, up) popisuje časovanou podmínku přechodu. Další forma výrazu vyskytujícím se v popise gramatiky je spouštěcí podmínka přechodu.

Na základě již předem definované relace částečného uspořádání na množině $\mathcal{M}$ je možno definovat tři následující druhy modu:

- primitivní mode - platí pro $M_i \in \mathcal{M}$ právě tehdy a jen právě tehdy když $\forall j M_j \not\sqsubseteq M_i$.
- složený mode - platí pro $M_i \in \mathcal{M}$ právě tehdy a jen právě tehdy když $\exists j M_j \sqsubseteq M_i$.
- kořenový mode - platí pro $M_i \in \mathcal{M}$ právě tehdy a jen právě tehdy když $\forall j M_i \sqsubseteq M_j$.

Z důvodu nutnosti analýzy hierarchie stromu modechartu je zavedena relace children: $\mathcal{M} \rightarrow 2^{\mathcal{M}}$, která je definována takto:

$$children(m) = \{n \mid n \sqsubset m \vee \nexists n' :: n \sqsubset n' \sqsubset m\}$$

Ikdyž relace $\sqsubset$ zachycuje stromovou hierarchii synů (resp. rodič), můžeme relaci children použít pro analýzu bezprostředních příbuzných modu. Z tohoto důvodu na základě této relace definujeme dva druhy vazby:

- bezprostřední syn, když $n \in children(m)$.
- bezprostřední rodič - z bodu výše je pro mode n bezprostředním rodičem mode m.

Jelikož Modechart je v jistém slova smyslu přechodovým systémem, je nutno specifikovat jakým způsobem je uchovávána informace o stavu takovéhoto systému. Dále je nutno uvést i způsob zachycení evolučního běhu daného systému. Pro účel zjištění aktuálního stavu systému je zaveden pojem globální mody systému. Uvažme $m \in \mathcal{M}$. Položme, že $G(m)$ specifikuje množinu globálních modu m. Dále pak označme $G_0(m)$ počátečním modem m. Stavová informace Modechart specifikace je dána $G(m_r)$, kde $m_r \in \mathcal{M}$ a je kořenovým modem. Másledně lze poté definovat vztahy pro:

- primitivní mode - platí $G(m) = \{\{m\}\}$ a $G_0(m) = \{\{m\}\}$.

- sériový mode - platí $G(m) = \bigcup_{m' \in children(m)} G(m')$ a $G_0(m) = G_0(initial(m))$.
- paralelní mode - poté $s \in G(m)$ právě a jen právě tehdy když $s = \bigcup_{m' \in children(m)} s_{m'}$ kde $s_{m'} \in G(m')$ pro všechny $m' \in children(m) \in G(m)$. $G_0(m) = \bigcup_{m' \in children(m)} G_0(m')$

Pro účel zachycení evolučního chování systému je zavedena relace Reach: $G \times \mathcal{T} \to G$, pro kterou uvažme následující. Máme přechod $\gamma \in \mathcal{T}$ tvaru (m_1, m_2, e) a položme, že $m_a \in \mathcal{M}$ je následníkem m_1 a $m_b \in \mathcal{M}$ je následníkem m_2. Dále uvažujeme $s, s' \in G$. Poté následný globální mode s' z s(je aktivní) za použití přechodu δ bude $s' = Reach(s, \delta)$. Tuto operaci lze obecně popsat takto:

- Odstraň globální mode m_a z s.
- Přidej $G_0(m_b)$
- Pro každého následníka m_i modu m_2 počínaje m_b, pokud m_i je seriovým modem, poté zaměň $G_0(m_i)$ za $G_0(m_{i+1})$

Je nutno zdůraznit, že relace Reach ukládá do globálního modu jen informaci o modech na nejnižní možné urovni. Je ale zcela zřejmé, že pokud je systém v nějakém předem daném modu, nachází se i současně ve všech rodičovských modech(toto globální mod explicitně nevystihuje). Poté po každém přechodu může docházet ke značným změnám. Proto je nutno zavést vztah pro vstup a výstup z modu a to jak explicitní, tak implicitní ($m, m' \in \mathcal{M}$:

Přechod explicitní vstupuje do modu m když:

- šipka přechodu končí na m nebo
- šipka přechodu protíná obdelník modu m a vchází do něj.

Přechod implicitně vstupuje do modu m když:

- m je počáteční mode a přímý rodič m' modu m je seriovým modem:
  - šipka přechodu končí na m' nebo
  - přechod implicitně vchází do m'.
- m' je přímý rodič m a je paralelním modem:
  - šipka přechodu končí na m' nebo
  - přechod implicitně vstupuje do m' nebo
  - přechod explicitně vstupuje do sourozence modu m.

Přechod explicitní opouští mode m když:

- šipka přechodu vychází z m nebo
- šipka přechodu protíná obdelník modu m a vychází z něj ven.

Přechod implicitně opouští mode m když:

- seriový mode m' je přímým rodičem m a systém je v modu m:
  - šipka vyhází z m' nebo
  - přechod implicitně opouští m'.

- mode m' je přímým rodičem m a systém je v modu m a m' je paralelní:
  - šipka vychází z m' nebo
  - přechod implicitně opouští m' nebo
  - přechod explicitně opouští sourozence modu m.

Je nutno poznamenat, že formalismus Modechart nezabraňuje vyvstání dvou událostí v jednou časovém okamžiku. Proto je nutno specifikovat podmínku pro výlučné opuštění modu, tak aby v danou chvíli mohl být veden jen jeden z přechodů. Výsledná podmínka nabývá formu:

$$\forall i \forall j @(e_1,\ i) \neq @(e_2,\ j).$$

Lub (least upper bound) pro $m, n \in \mathcal{M}$ (s ohledem na relaci $\sqsubset$) je definováno:

$$lub(m,n) = \{l \mid m \sqsubset l \wedge n \sqsubset l \wedge \nexists l' :: m \sqsubset l' \wedge n \sqsubset l' \wedge l' \sqsubset l\}$$

Na základě specifikované informace lze definovat pro $m, n \in \mathcal{M}$ souběžnost zapisovanou ve formě $m \,||n$. Ta platí právě tehdy a jen tehdy když $m \not\sqsubset n \wedge n \not\sqsubset m \wedge lub(m,n) \in P$. Dva mody $m, n \in \mathcal{M}$ jsou sekvenční, definováno $m \sim n$, právě tehdy a jen právě tehdy když $m \not\sqsubset n \wedge n \not\sqsubset m \Rightarrow m \sim n \equiv \neq (m||n)$. Poté pro dva mody $m, n \in \mathcal{M}$, $m \neq n$ platí:

$$m \not\sqsubset n \wedge n \not\sqsubset m \Rightarrow m \sim n \equiv \neg(m||n)$$

Na základě nastíněných definic lze definovat, kdy je model specifikovaný Modechartem správný a syntakticky korektní. Nutnými podmínkami jsou tyto:

- $\forall s \in S :: \exists m \in children(s) :: initial(s) = m$. Slovně tato podmínka specifikuje, že každý seriový mode musí mít počáteční mode.
- $\forall (s,t,e) \in \mathcal{T} :: s \sim t$. Podmínka popisuje fakt, že přechody moho být vedeny jen mezi sekvenčními mody.

Tímto byl shrnuta formální definice Modechart a notace, které budou dále použity ve zbývajícím textu. Důkladné pochopení popsané problematiky je nutné pro pochopení následující části popisující přístup k analýze běhu systému založeného na specifikaci formalismem Modechart.

4.1.2 Přístup k analýze

Cílem je použít Modechart specifikaci systému a verifikovat vlastnosti s ohledem na konečný počet výskytu událostí. Budou definovány výpočetní posloupnosti systému s použitím orientovaného stromu, který bude rozšířen o množinu relací popisujících časovou separaci jednotlivých událostí. Tato separace bude representována spodní a horní hranicí, resp. zpožděním a nejzažší mezí. Pokud jsme schopni representovat potenciálně nekonečný výpočetní strom pomocí konečného grafu, poté rozhodovací procedůra je schopna ověřit vlastnosti systému.

Obrázek 4.1 ukazuje plánovaný přístup. Modechart specifikace je použita pro účel generování konečného výpočetního grafu. Cílem je vytvořit graf takový, aby plně reflektoval chování popsaného systému. Na rozdíl od modelu modechart specifikace, který přiřazuje čas událostem, je výpočetní graf prostředkem přiřazujícím čas událostem na cestě v grafu. Pokud použijeme vhodnou proceduru převodu odpovídá výpočetní model výchozímu modelu systému a opačně. Poté lze položit, že pokud rozhodneme o vlastnosti systému na základě výpočetního grafu, bude tato skutečnost platit i pro Modechart model samotný.

Obrázek 4.1: Schéma ověřování vlastností.

Ještě než bude přikročeno k popisu algoritmu pro převod Modechart specifikace do podoby grafového modelu, je nutno přiblížit některé základní pojmy a definice. Systém popsaný pomocí Modechart specifikace budeme převádět do podoby dvou matematických struktur. První z nich je již zmíněný výpočetní strom, resp. jeho konečná podoba ve formě grafu a dále zde vystupuje separační graf. První z nich popisuje evoluční chování systému, které je řízeno na základě struktury separačního grafu, jenž obsahuje časové vzdálenosti jednotlivých událostí a definuje časová omezení.

Nyní přikročíme k definici výpočetního stromu jako matematické struktury popisující běh systému. Každá cesta z kořenového uzlu tohoto stromu definuje časovanou výpočetní trajektorii systému. Každy uzel stromu(v následujícím textu dále nazývaný jako bod stromu) představuje konfiguraci systému popsaného pomocí Modechartu a musí obsahovat dostatek informací o stavu systému aby bylo možno zjistit, které mody jsou aktivní a jaké přechody mohou být v daném čase vedeny.

Výpočetní strom je orientovaný acyklický graf a je definován pomocí množiny uzlů V a množiny hran E. Množina hran je ve tvaru $V \times V$. Je kladeno omezení aby prvky této relace netvořila cykly a to ani reflexivní. Dále je definována relace λ, která přiřazuje každému uzlu označení ve tvaru klíčových informací o něm. Tato relace definuje následující tři aspekty:

- jméno bodu
- vstupní událost
- množinu událostí

Jméno bodu odpovídá seznamu všech modu, které jsou aktivní v daném okamžiku výpočetní trajektorie. Je zde použita relace G definována v předešlé kapitole, která je aplikována na kořenový mode. Pro kořenový bod stromu je jeho název definován jako $G_0(m)$, kde m je kořenovým uzlem a relace G_0 je chápána ve smyslu zavedeném dříve.

Vstupní událost popisuje událost, která způsobila vstup do daného bodu stromu. Pro kořenový bod stromu se jedná o počáteční událost výpočtu. Pro ostatní body je tato veličina definována přechodovou událostí nějakého přechodu Modechart specifikace, která způsobila přechod systému do daného bodu výpočetní trajektorie.

Množina událostí je podmnožinou množiny podmíněných událostí(definováno dále). Účelem existence této množiny je skutečnost, že na jejim základě můžeme rozhodovat o splnění podmínek spouštěného přechodu pro daný bod. Události této množiny jsou definovány v čase vstupu do daného bodu a může se jednat o událost popisující opuštění předešlého modu, vstupu do aktuálního modu nebo vedení přechodu. Tato množina je podmíněna omezením pro dědění událostí. Tímto způsobem mohou být při splnění určitých podmínek poděděny události předešlého bodu trajektorie.

Výše vzpomenutá množina podmíněných událostí obsahuje všechny události takové, že jsou součástí spouštěcích podmínek jakéhokoliv přechodu daného modechart modelu. Toto nám umožní

ukládat a dále i ve výpočetním stromu uvažovat jen události, které dávají smysl a podmiňují nějaký námi uvažovaný přechod.

Na základě již definovaného pojmu bodu trajektorie(resp. uzlu výpočetního stromu) definujeme nečasovanou trajektorii výpočtu. Trajektorií modechart specifikace chápeme sekvenci $\{\sigma_i\}_{i\geq 0}$ bodů specifikace takových, že σ_0 je počátečním bodem a pro všechny další body σ_i musí platit, že jejich vstupní událostí je přechodová událost nějakého přechodu, který je proveditelný v σ_{i-1}. Množina události bodu poté obsahuje podmíněné události vyvstalé vedením přechodu a nebo sjednocení událostí vyvstalých vedením přechodu a množiny událostí bodu σ_{i-1}. V druhém případě hovoříme o tom, že bod σ_i dědí události bodu σ_{i-1}. Přechod je proveditelný v daném bodu trajektorie σ_i, pokud počáteční bod je součástí jména bodu a jedná se o časovaný přechod nebo o přechod se spouštěcí podmínkou, která je splněna pro množinu událostí daného bodu. Je nutno dále konstatovat, že jména bodů σ_i a σ_{i+1} jsou definovány na základě již dříve specifikovaných podmínek pro explicitní/implicítní vstup či opuštění modu. Pokud dojde k porušení těchto podmínek, nejedná se poté o výpočetní trajektorii Modechart modelu. Počáteční bod σ_0 má jmeno specifikované relací G_0 nad kořenovým modem specifikace. Je zde definována množina událostí, která odpovídá počáteční množině událostí systému.

Definice výše nebrala ohled na časový charakter přechodů mezi běhy trajektorie. Proto bude zaveden pojem omezená trajektorie. Jedná se o trajektorii, jejíž každé dva body jsou rozšířeny čas a časová omezení. Dále je brán zřetel na časovou separaci a dědění množiny událostí. Časová omezení nad omezenou trajektorii jsou definovány takto(relace time přiřazuje každé události nenulový čas):

1. $time(\sigma_i) \geq 0$
2. $time(\sigma_i) \leq time(\sigma_{i+1})$
3. Pokud σ_{i+1} dědí množinu událostí bodu σ_i, poté $time(\sigma_{i+1}) = time(\sigma_i)$
4. Pokud σ_{i+1} nedědí množinu událostí bodu σ_i, poté $time(\sigma_i) + 1 \leq time(\sigma_{i+1})$
5. Pokud vstupní událostí σ_{i+1} je přechodová událost pro přechod řízený spouštěcí podmínkou, poté $time(\sigma_{i+1}) = time(\sigma_i)$
6. Pokud vstupní událostí σ_{i+1} je přechodová událost pro přechod řízený časovým omezením (r, d) a σ_j je referenčním bodem přechodu v bodu σ_i, poté $time(\sigma_j)+r \leq time(\sigma_i)$ a $time(\sigma_i)-d \leq time(\sigma_j)$
7. Pokud je přechod řízený spouštěcí podmínkou proveditelný v σ_i potom $time(\sigma_{i+1}) = time(\sigma_i)$.
8. Pokud v σ_i existuje přechod podmíněný časovou podmínkou v horní časovou mezí d a σ_j je jeho referenčním bodem, potom $time(\sigma_i) - d \leq time(\sigma_j)$

Podmínky v bodě 6. a 7. používají pojem referenční bod. Jedná se o bod, oproti němuž je měřeno časové omezení. U přechodu na základě spouštěcí podmínky je jedná o předcházející bod. Pro přechod řízený časovou podmínkou uvažme, že jeho zdrojový mode je aktivní v σ_k a také po celou dobu až do σ_i, kde $k \geq i$. Poté je σ_i referenčním bodem časovaného přechodu proveditelného v σ_k.

Body 1. a 2. zajistí, že napříč omezenou trajektorií nesmí čas klesat. Dále další dvě podmínky popisují korektně, kdy lze dědit z předešlého bodu množinu událostí. Bod 5. zajistí, že přechod na základě spouštěcí události je veden hned v okamžiku, kdy je daná podmínka naplněna. Další bod popisuje definování spodní a horní časové hranice časovaného přechodu a omezení nad nimi. Poslední dvě podmínky popisují nemožnost volení přechodu, který by narušil časová omezení.

Kompletní omezená trajektorie je omezenou trajektorií, která je buď nekonečná, nemá v koncovém bodě žádné proveditelné přechody nebo v konečném bodě má přechod s časovanou podmínkou a horní nekonečnou časovou hranicí.

Výpočet Modechart modelu je přiřazením času bodům kompletní omezené trajektorie tak, aby nedošlo k porušení časových omezení.

Je nutno specifikovat dva velice důležité pojmy, jimiž jsou možný a skutečný následník. Bod je možným následníkem tehdy, pokud jeho přidáním na konec trajektorie získáváme také trajektorii. Možný následník je bodem, který může být volen z daného bodu σ_i bez ohledu na časová omezení. Skutečný následník je možným následníkem a přidáním tohoto bodu na konec kompletní omezené trajektorie získáme znovu kompletní omezenou trajektorii. Jinými slovy přidáním tohoto bodu nebudou porušeny žádné kladená kladená časová omezení.

Kompletní omezená trajektorie má výpočet právě tehdy a jen právě tehdy když každý bod (jiný než bod počáteční) je skutečným následníkem svého předchůdce.

S ohledem na předešlé definice můžeme výpočet systému specifikovaného Modechartem chápat jako výpočetní strom, kterého uzly jsou body a potomci bodů jsou jejich skuteční následníci. Dále trajektorie bude každou cestou v tomto stromu.

Množina časových omezení modelu je representována pomocí orientovaného grafu označovaného jako separační graf. Uzly reprezentují přechodové události bodů výpočetního stromu. Ohodnocené hrany na druhou stranu popisujíčasové separace mezi uzly (a jsou omezeními odpovídající omezené trajektorie). Dále aby nedošlo k nejednoznačnostem oproti definici časových omezení, tak její body 7. a 8., které zavádějí horní časovou hranici přechodu nejsou uvažovány, pokud přechod je proveditelný a může být a je volen později. Konstrukce a princip separačního grafu je vcelku jednoduchý. Uvažujme trajektorii $P = (e_0 = e_r, v_0 = r, e_1, v_1, ..., e_n, v_n)$. Separační graf $SEP(P)$ pro P se skládá z uzlů e_i^* pro každou hranu e_i trajektorie. Hrany jsou definovány následovně:

- Je vedena hrana s váhou z e_i^* do e_{i+1}^*. pro všechny $0 \leq i < n$. Tato hrana kopíruje a zachovává stromovou strukturu grafu.
- Pokud e_i je referenčním bodem pro časovaný přechod proveditelný v e_j s časovými ohraničeními (r, d), přidáme hranu s váhou r z e_i^* do e_j^* pokud $r > 0$. Vedeme také hranu s váhou $-d$ z e_j^* do e_i^* pokud $d \neq \infty$.
- Pro přechod omezený spouštěcí podmínkou přidáváme hranu s váhou 0 vedoucí z e_j^* do e_{j-1}^*.

Je nutno poznamenat, že separační graf neobsahuje pozitivní cykly. Pokud by tomu tak bylo, došlo k porušení časových omezení systému.

Pro účel následné analýzy separačního grafu je zavedena relace *distance*. Zápis $distance(A, B)$ popisuje separační vzdálenost uzlů A a B. Tato vzdálenost je definována jako maximálně ohodnocená cesta v grafu mezi uzly A a B. Pokud takováto cesta vůbec neexituje $distance(A, B) = -\infty$. Zvolme libovolné dva body A a B na cestě výpočetního stromu kde A předchází B, poté:

- $distance(A, B) \geq 0$ a tato hodnota udává, kdy lze nejdříve volit B relativně k A.
- $distance(B, A) \leq 0$ což udává, kdy lze nejpozději volit B vzhledem k A.

Tato relace pomůže v definici způsobu výpočku skutečného následníka bodu výpočetního stromu. Postup výpočtu je založen na zkoumání vzdáleností mezi možnými následníky, z nichž se poté vybere skutečný následník(pokud existuje) a to následovně:

1. Uvažujme uzel výpočetního stromu a množinu jeho možných následníků.

2. Pro každého možného následníka přidej dočasně do separačního grafu uzel.

3. Zvol možného následníka A_i a opakuj pro všechny. Pokud platí pro všechny ostatní možné následníky A_j vztah $distance(A_j, A_i) \leq 0$, prohlaš za skutečného následníka, jinak vymaž.

S ohledem na to, že již byly definovány všechny potřebné zákonitosti, můžeme přistoupit k definici algoritmu konstrukce výpočetního stromu. V dalším textu dále ukážeme, jakým způsobem lze dosáhnou následné konstrukce výpočetního grafu. Algoritmus, který zajistí převod má následující tvar:

```
Algoritmus konstrukce výpočetního stromu
VSTUP: modechart M

1. Vytvoř kořen r stromu;
2. r.global_mode = počáteční global mode M;
3. r.events = spouštěcí události v˜době startu;
4. Přidej vstupující hranu e_r do r
5. for každý list v stromu do
6.    for každý aktivní mode m v˜v.global_mode do
7.    for každý přechod γ z˜m do
8.       if γ je časovaný přechod nebo
            spouštěcí podmínka je splněna v˜v
9.          Generuj možného následníka v' listu v;
10.         v'.global_mode = Reach(v.global_mode, γ);
11.         v'.events = spouštěcí podmínka genrovaná γ
12.         (v, v').transition = γ;
         endif
      endfor
      endfor
13.   Spočti množinu skutečných následníků listu v; Odstraň ostatní;
14.   if v.events ≠ {}
15.   for každého skutečného následníka v' do
16.      if (v, v') musí nastat okamžitě po vstupu do v
17.         Přidej v.events do v'.events;
18.      elseif může nastat okamžitě nebo později
19.         v'' = kopie v';
20.         Přidej v.events do v'.events;
         endif
      endfor
      endif
   endfor
end.
```

Zápis algoritmu sice vypadá relativně složitě ale základní myšlenka je vcelku jednoduchá. Je vytvořen kořenový uzel stromu, který má globální mode roven počátečnímu globálnímu modu systému. Do množiny událostí tohoto uzlu jsou nastaveny všechny události platné v počátečním stavu. Poté algortimus iteruje smyčku, která vždy volí listový uzel, který se dále pokouší rozšíří o skutečného následníka(resp. následníky). Pro listový uzel je zkoumán jeho globální mode a je zjištěno, které přechody jsou v daném okamžiku splnitelné. Na základě těchto přechodů je

generována množina možných následníků. Jejich globální mode je dán již dříve popsanou relací *Reach* a množina událostí je generována zavedeným způsobem. Pokud jsem byli schopni určit množinu možných následníků, můžeme již výše nastíněným způsobem vybudovat množinu skutečných následníků. Prvky této množiny jsou poté vloženy do vytvářeného stromu a je pro ně ponechán i uzel v separačním grafu. Dále na základě popsaných pravidel pro omezenou trajektorii je nutno při nulové separaci dvou událostí zajistit dědění množiny událostí z předka tak, aby byly dodrženy podmínky uspokojení spouštěných přechodů.

Byla popsána konstrukce výpočetního stromu, který reflektuje kompletní omezenou výpočetní trajektorii specifikace Modechart modelu. Je nutno demonstrovat a zavést způsob, jakým je možno převést tuto strukturu do podoby grafu s konečným počtem uzlů. Toto nám poté umožní použití jistých postupů umožňujících ověžení logických tvrzení vůči takovémuto typu popisu. Na základě zkoumání výpočetních stromů bylo zjištěno, že některé podstromy vykazují podobné vlastnosti. Byly definovány techniky, které dokáží najít uzly takové, že jejich podstromy jsou vzájemně isomorfní. Toto dále umožňuje takovéto uzly sloučit a postupně vytvářet konečnou podobu ve formě výpočetního grafu. Dále bylo dokázáno, že pro každý výpočetní strom existuje po aplikaci detekce isomorfismu jeho konečná grafová representace. Abychom mohli položit, že dva uzly a jejich podstromy jsou isomorfní, musí být splněno pro uzly v a w následující:

1. Oba uzly musí mít stejný globální mode.
2. Množina událostí musí být u uzlů v a w totožná.
3. Pro uzly v a w musí být splněna deq podmínka(distance equivalence condition)

Deq podmínka říká, že pokud máme uzly v a w a jejich množinu možných následníků Δ resp. Δ', musí pro každé $A \in \Delta$ a $B \in \Delta'$ platit:

$$\begin{aligned} distance(A, B) &= distance(A', B') \\ distance(B, A) &= distance(B', A') \end{aligned}$$

Pokud pro zmíněné uzly v a w platí všechny tři zmíněné podmínky, můžeme prohlásit jak uzly samotné, tak jejich podstromy za isomorfní. Poté jsme schopni dané dva uzly sloučit a to tak, že z přímého předchůdce uzlu w vedeme hranu přímo do v. Již podle předešlé definice je tímto způsobem realizovatelný převod potenciálně nekonečného výpočetního stromu do podoby konečného grafu.

V této podkapitole byl přiblížen výpočet specifikace Modechart modelu. Byla definována trajektorie výpočtu a dále i omezená trajektorie výpočtu, která dodržuje časová omezení systému. Následně byla dodefinována kompletní omezená trajektorie, která je reflektována výpočetním stromem systému. Nastiněn byl obecný algortimus převodu modelu do formy výpočetního stromu a dále bylo ukázáno, jak konstruovat separační graf a jakým způsobem lze na jeho základě rozhodnout o skutečných následnících uzlu. Popsán byl postup, který dovolí na základě analýzy isomorfismu podstromů výpočetního grafu vytvořit jeho konečnou grafovou representaci. Tímto jsme schopni vybudovat odpovídající výpočetní model Modechart z jeho specifikace a v následující podkapitole bodou ukázány postupy, jakými lze ověřit platnost tvrzení vůči tomuto modelu.

4.1.3 Analýza vlastností

V předešlé podkapitole bylo popsáno, jakým způsobem lze převést výpočetní trajektorii modelu specifikovaného pomocí Modechart do podoby výpočetního grafu. Bylo položeno, že tyto dva modely jsou ve své podstatě ekvivalentní. Jakákoliv vlastnost, která je platná ve výpočetním grafu je pravdivá i pro zdrojový Modechart model. V této podkapitole bude nastíněno jaké vlastnosti lze ověřit oproti výpočetnímu grafu.

Dále popíšeme dvě třídy časových vlastností popsaných pomocí RTL pro které existují jednoduché postupy ověření jejich platnosti vůči výpočetnímu grafu. Je nejprve nutno zavést určité pojmy a definice.

Koncový bod(endpoint) je dán aplikací výskytové funkce tvaru $@(E, i \pm k)$, kde E odpovídá události, i je celočíselnou proměnnou a c je nezáporné celé číslo.

Dva body jsou související, pokud oba obsahují stejnou celočíselnou proměnnou v indexu výskytové funkce. Poté $@(E_1, x)$ a $@(E_2, x + 8)$ jsou souvisejícími body. Ale například $@(\downarrow A, i)$ a $@(\downarrow A, j)$ spolu nesouvisí.

Pokud uvažujeme dvě události E_1 a E_2, můžeme na jejich základě definovat interval. Položme, že k_1 a k_2 jsou nezáporná celá čísla a i jeceločíselná proměnná. Poté interval chápeme jako dvojici souvisejících bodů tvaru: $@(E_1, i \pm k_1)$ a $@(E_2, i \pm k_2)$

Uvažujme výpočetní graf G a dvojici událostí souvisejících koncových bodů E_1 a E_2. Cyklus grafu zachovává dvojici koncových bodů pokud počet bodů označených E_1 je roven počtu bodů označených E_2. Následně výpočetní graf G zachovává RTL formuli pokud každý cyklus grafu zachovává každou dvojici souvisejících koncových bodů.

V polynomiálním čase lze zjistit, jestli dva koncové body jsou zachovány. Výpočet zachovává dvojici koncových bodů obsahujících události E_1 a E_2 pokud platí jakýkoliv bod níže:

1. $E_1 = E_2$, což znamená, že se jedná o stejné události.
2. $E_1 =\rightarrow M$ a $E_2 = M \rightarrow$, jenž popisuje, že události jsou vstupní a výstupní události stejného modu.
3. $E_1 = (S := true)$ a $E_1 = (S := false)$, což předpokládá, že tyto události jsou přechodovými událostmi nad stejnou proměnnou.
4. $E_1 =\uparrow A$ a $E_2 =\downarrow A$, jenž popisuje, že dané události jsou počáteční a koncovou událostí stejné akce.

Následně budou popsány dvě třídy RTL formulí, pro které existuje rozhodovací procedura, která určí splnitelnost resp. nesplnitelnost.

Minimální a maximální vzdálenost mezi koncovými body

Třida časových omezení specifikuje relativní a absolutní uspořádání, tak časovou vzdálenost mezi souvisejícími koncovými body. Uvažujme dva související koncové body e_1 a e_2 a nezáporné celé číslo k udávající časovou vzdálenost. RTL formule popisující vlastnost je ve tvaru $\exists xF$ nebo $\forall xF$, kde F je formule bez kvantifikátorů s nerovnostmi tvaru:

$$e_1 \pm k \leq e_2.$$

Každá tato formule může být převedena do tvaru, který specifikuje:

1. Minimální časovou vzdálenost k mezi koncovými body: $k \leq e_2 - e_1$.

2. Maximální časovou vzdálenost k mezi koncovými body: $e_2 - e_1 \leq k$.

Uvažujme, že G je výpočetním grafem a F je RTL formule specifikující minimální/maximální vzdálenost mezi koncovými body. Níže jsou pro tyto dvě třídy uvedeny algoritmy, které daný problém rozhodnou. Pokud F je univerzálně kvantifikována, musíme spustit příslušný algoritmus pro všechny body označení E_j v grafu G a musí být vždy vráceno *true* aby celá formule byla pravdivá. Jinak pokud je formule F kvantifikována existenčně stačí aby při nejmenším jeden běh odpovídajícího algoritmu nad bodem označeným E_j vrátil *true* pro graf G aby byla daná formule splněna.
Rozhodovací procedury pro dané dvě třídy problémů mají odpovídající tvar:

```
1. procedure check_min_distance(e_1, e_2, k, G, F)
2.   najdi bod P označený E_j, kde 1 ≤ j ≤ n;
3.   najdi všechny odpovídající koncové body F rozvinutím každého
     cyklu konstantní počet krát;
4.   d := nejkratší vzdálenost mezi e_1 a e_2 v˜rozvinutém grafu G;
5.   if k˜ ≤ d poté return true; else return false;
```

```
1. procedure check_max_distance(e_1, e_2, k, G, F)
2.   najdi bod P označený E_j, kde 1 ≤ j ≤ n;
3.   najdi všechny odpovídající koncové body F rozvinutím každého
     cyklu konstantní počet krát;
4.   d := nejdelší vzdálenost mezi e_1 a e_2 v˜rozvinutém grafu G;
5.   if d ≤ k poté return true; else return false;
```

Exkluze a inkluze koncového bodu a intervalu

Tato třída časových vlastností specifikuje inkluzi nebo exkluzi koncového bodu nebo intervalu v rámci jiného intervalu. K výrazu přidáváme celočíselný offset a vznikne tvar: $@(E, i \pm k) \pm c$.

Uvažujme interval I_1 s koncovými body e_1 a e_2 a další interval I_2 s koncovými body e_3 a e_4. RTL formule F této třídy vlastností systému je ve formě:

$$Q_a Q_b e_1 \leq e_3 \wedge e_4 \leq e_2, \text{ kde } Q_a \text{ a } Q_b \text{ jsou kvantifikátory pro index výskytu.}$$

Zde je uvažováno, že iterval I_1 obsahuje interval I_2. Dalším případem je exkluze intervalu, což znamená, že daný interval neobsahuje jiný, zapsáno:

$$Q_a Q_b e_1 \leq e_3 \vee e_4 \leq e_2, \text{ kde } Q_a \text{ a } Q_b \text{ jsou kvantifikátory pro index výskytu.}$$

Typ algoritmu, který rozhodne pravdivost na základě výpočetního grafu a RTL formule specifikující časovou závislost záleží na druhu kombinaci kvantifikátorů Q_a a Q_b. Také je zde nutno brát v úvahu jestli se jedná o operaci inkluze nebo exkluze intervalu. Uveďme příklad, že požadujeme ověřit jestli nějaký interval I_1 obsahuje instanci intervalu I_2. Nejprve musíme v grafu identifikovat instanci intervalu I_2. Dále zjistíme, jestli můžeme najít interval I_1, takový, že obsahuje I_2. Pokud tomuto tak je, vrací rozhodovací procedura hodnotu *true*. Pokud bychom chtěli zjistit, jestli všechny intervaly I_1 obsahují instanci I_2, museli bychom opakovat zmíněný postup pro každy nalezený interval I_2. Další možné kombinace kvantifikátorů mohou být ověřeny velmi podobným způsobem.

Další třídy zjišťování vlastností

Na základě popsaných dvou tříd lze použít jejich kombinaci pro specifikaci dalších rozšíření. Tyto vlastnosti nemohou být vyjádřeny žádnou z dvou dříve popsaných tříd bez dalších rozšíření. Uvažujme dva případy:

- Každá instance intervalu α obsahuje instanci intervalu β a γ.
- Tři intervaly α, β a γ obsahují každy další ve výčtu. Čili interval α obsahuje interval β a dále tento obsahuje interval γ.

4.2 Rozhraní a funkcionality systému

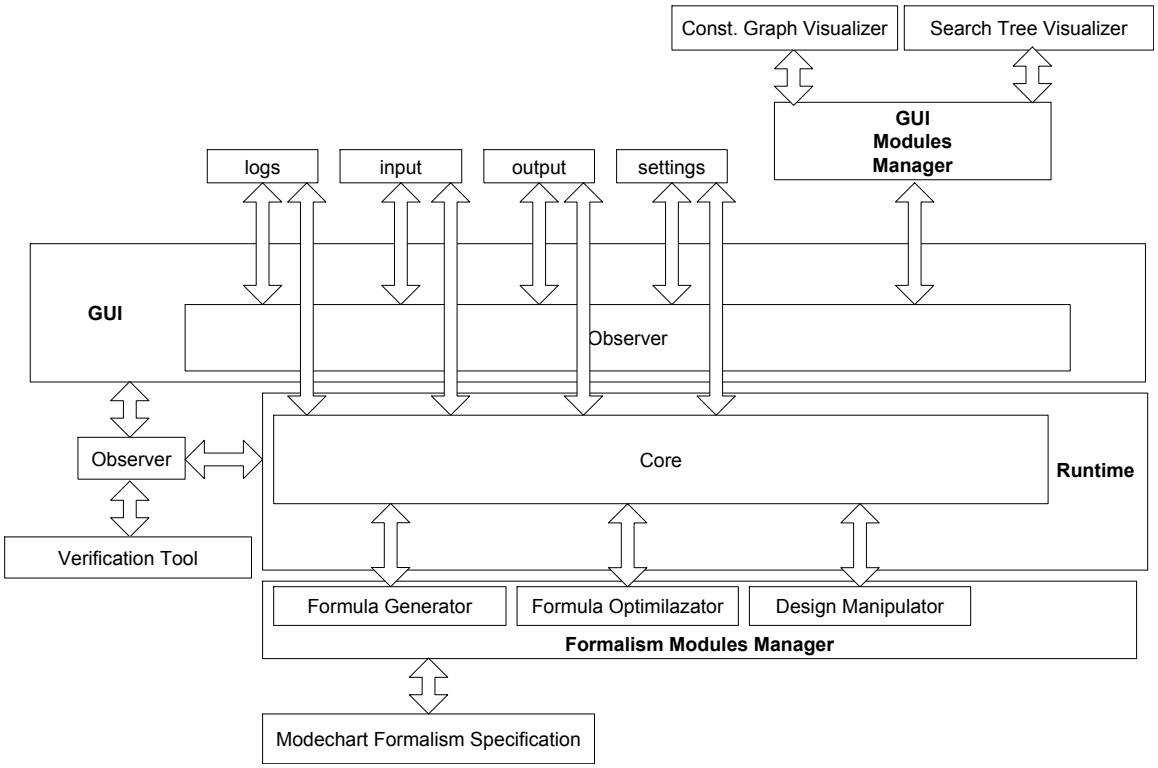
Tato kapitola je zaměřena na popsání navrhované struktury programu. Byly brány v úvahu všechny požadavky a omezení, která mají být kladena na výsledný systém. Navrhované řešení je plně modulární a řídí se klasickými koncepty objektově orientovaného přístupu k analýze a návrhu systému.

4.2.1 Definice blokového schématu programu

Schéma na obrázku 4.2 popisuje navrhovanou strukturu programu. Cílem je od začátku vytvořit univerzální rozhraní pro specifikaci a následnou verifikaci systémů. Je kladen důraz na to aby výsledný program nebyl závislý na použitém formalismu popisu. Tudíž aby na modulární bázi dovoloval vytvořit jakýkoliv modul implementující uživatelem definované metodiky popisu. Dále se specifikace zaměřuje na to aby bylo možno do výsledného grafického rozhraní vkládat jakékoliv moduly realizující různé druhy vizualizace výsledků. Tyto moduly by vhodně vyhodnocovaly a zobrazovaly výsledky verifikace poskytované zvoleným nástrojem pro ověřování korektnosti systému. Jednalo by se například o generování statistik, vytváření grafů průběhu, grafů omezení systému, stromů běhu systému, stromů volání funkcí ap. Systém by neměl být závislý na zvoleném nástroji pro následnou verifikaci. Proto je nutno aby bylo definováno vhodné rozhraní pro komunikaci, nad kterým již bude možné vybudovat jakýkoliv komunikační protokol. Jako například síťový, objektový nebo založený na sdílené paměti.

4.2.2 Popis základních bloků a funkcionalit

Systém definuje dvá klíčové bloky programu jimiž je komponenta řídící celé grafické uživatelské rozhraní a poté samotné jádro systému. Jádro obstarává operace na nejnižší úrovni. Provádí komunikaci pomocí vhodně navrženého rozhraní s verifikačním nástrojem. Dále ma na starost manipulaci s formulemi popisujícími navrhovaný systém. Jedná se jmenovitě o principy umožňující generování formulí z popisu systému definovaného pomicí daného formalismu. Tyto formule mohou být poté vhodně optimalizovány různými druhy technik. Specifikovány by zde měly být také způsoby umožňující praci s popisem verifikovaného systému a to na nejnižší úrovní. Jmenovitě v závislosti na povaze zvoleného formalismu popisu se může jednat např. o práci se stromy, seznamy, poli binárních vektorů atd. Uživatel je schopen modulárně naprogramovat a vložit do programu jakýkoliv vhodný formalismus pro popis. Podmínkou je aby konkrétně implementoval algoritmy pro již dříve zmíněné tři typy operací. A poté musí nabídnout postupy schopné definovat, měnit a zobrazovat grafický model systému. Jádro je stejně jako modul pro GUI schopno využívat operace pro vstup a výstup, vypisovat ladící informace a přistupovat ke globálnímu nastavení programu.



Obrázek 4.2: Blokové schéma programu.

Komponenta řídící grafické uživatelské rozhraní je propojena s verifikačním nástrojem od kterého dostává informace o průběhu a výsledku verifikace. Tyto informace jsou dále směřovány do rozhraní systému pro správu modulů, který je schopen obsluhovat jakýkoliv druh uživatelem specifikovaných grafických komponent. Tyto prvky poté vhodně zobrazují dosažené výsledky.

Kapitola 5

Implementace aplikace

5.1 Modularita a rozšiřitelnost

Základem každé moderní aplikace je to, že dodržuje zavedené metodiky objektově orientovaného návrhu. Mezi základní části návrhu a implementace jakékoliv aplikace by mělo patřit vyhodnocení modularity a následné rozšířitelnosti.

Pod pojmem modularity rozumíme rozdělení aplikace do logických částí, které jsou mezi sebou minimálně závislé. Toto umožní pohled na aplikaci z hlediska jejich dílčích částí a ne celku samotného. Nejčastějším přístupem, který řeší problematiku modularity je rozdělení aplikace do několika menších zásuvných modulů. Základním problémem je zde lokalizace zásuvných modulů, jejich načítání, propojení a případné řešení závislostí. Aby nebylo potřeba tyto postupy zdlouhavě implementovat, tak všechny zmíněné problémy by za nás měla řešit specializovaná aplikace. Jak bude naznačeno dále, k tomuto účelu je nejlepší volit některé z volně dostupných řešení.

Většina aplikací vzniká inkrementálně tak, že k základním funkcionalitám jsou dodávány nové, které rozšiřují její možnosti. Mira a flexibilita této operace je popsána pomocí rozšiřitelnosti. Pokud je produkt dostatečně rozšiřitelný, minimalizuje se vložené úsilí, a tím i vynaložené finanční prostředky. Je tudíž důležité, aby výsledná aplikace vhodně a včasně reagovala na změnu na ni kladených požadavků.

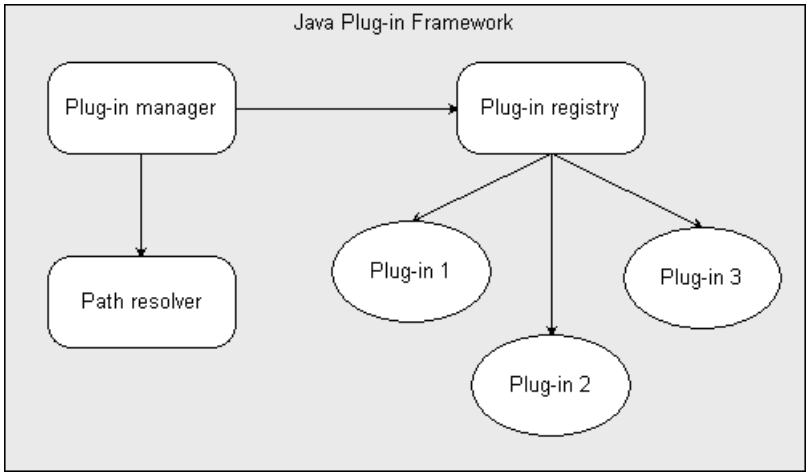
5.1.1 Java Plugin Framework

Pro potřeby programu byl zvolen volně dostupný produkt Java Plugin Framework. Bylo shledáno, že jako jedno z mála řešení nabízí kompromis mezi složitostí a poskytovanými funkcionalitami. Použití tohoto frameworku ve značné míře zvýšilo rozšiřitelnost a modularitu vytvářené aplikace. V konečném důsledku to znamená, že vytvoření jakékoliv další součásti nebo rozšíření stávající bude velice jednoduché a flexibilní.

JPF framework je založen na konceptu zásuvných modulů, které se snaží strukturovaně akumulovat kód a zdroje programu do jednoho uceleného celku. Shlukování kódu a dat je prováděno na základě společných a charakteristických rysů a je taky dbáno na to aby závislost na dalších zásuvných modulech byla minimální. Rozšiřitelnost zásuvných modulů o další funkcionality je zajištěna pomocí dvou konstrukcí: bodů rozšíření (extension point) a rozšíření. Jednoduše řečeno, je bod rozšíření místem zásuvného modulu, kde implicitně říkáme, že je možno připojit jiný rozšiřující modul. Rozšíření zase na druhou stranu popisuje mechanismus, kterým daný modul sděluje, že může nabízet implementaci jistých služeb. Jednoduchým příkladem může být modul, který provádí databázové operace a definuje bod rozšíření pro účel napojení modulů implementujících tříd pro databázovou komunikaci. Poté může existovat několik modulů, kde každý nabízí databázové spojení

na databazi jiného výrobce a tyto definují rozšíření. Poté aplikace může vyhledat a připojit několik různých implementací databázových spojení a je nutno poznamenat, že následně již není problém kdykoliv dodat implementaci pro další výrobce DB.

Framework samotný nedefinuje jen tyto součásti, ale musí poskytovat služby pro manipulaci a správu zásuvných modulů. Zásuvný modul samotný zase na druhou stranu musí poskytnout potřebné informace o sobě. Všechna klíčová data jsou uložena v XML souboru přímo spefikovaného formátu (pomocí DTD). Poskytnutých dat o modulu může bý opravdu mnoho, jelikož na jejich základě služby systému provádějí analýzu. Jmenovitě se může jednat o analýzu konzistence stromu zásuvných modulů. Zde dochází ke zkoumání závislostí mezi moduly a to nejenom na základě jména a typu modulu, ale také i jeho verze. Také dochází k ověření rozšíření, kde se kontroluje, jestli byly deklarovány včetně všech potřebných parametrů.



Obrázek 5.1: Schéma služeb JPF [6].

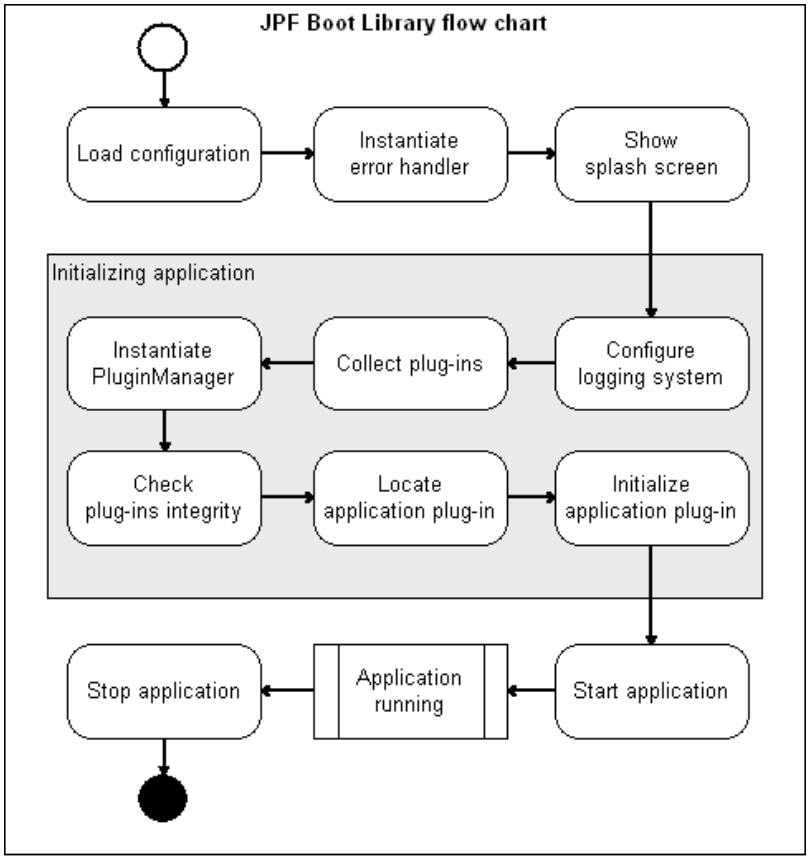
Na obrázku 5.1 je vidět základní schéma nabízených služeb. Je patrné, že kromě tří zásuvných modulů zde lze identifikovat další tři části a to jmenovitě:

- Registr zásuvných modulů - spravuje informace o všech nalezených modulech.
- Správce zásuvných modulů - služba načítající a aktivující zásuvné moduly.
- Správce umístění - je zodpovědný za fyzické nalezení všech zdrojů.

Registr zásuvných modulů se stará o všechna metadata nalezených modulů. Zásuvný modul je registrován tak, že jsou předány metainformace modulu specifikovány v již zmiňované formě XML souboru. Poté může být modul kdykoliv aktivován či deaktivován. Výhodou je, že tuto operaci lze provádět nejen při startu programu, ale i za jeho běhu. Tím pádem můžeme hovořit o tzv. hot plug-in. Správce modulů se pro změnu stará o fyzickou manipulaci. Umožňuje aktivaci modulu zavedením jeho inicializační třídy. Služba může manipulovat se všemi moduly, které mají svá metadata zavedená v systému. Dovoluje také spouštět a pozastavovat moduly za běhu, čímž redukuje množství programem alokované paměti.

Na obrázku 5.2 je ilustrován celý životní cyklus aplikace. Celý proces je relativně jednoduchý a lze jej popsat v několika krocích:

- načtení konfigurace aplikace



Obrázek 5.2: Schéma zavaděče JPF [6].

- shromáždění dostupných zásuvných modulů
- vytvořenín instance a inicializace běhového porstředí JPF
- publikování (zavedení) nalezených modulů
- aktivace a spuštění výchozího modulu
- při ukončení aplikace je potřeba korektně deinicializovat celý framework

JPF Framework je výhodný v tom, že jej bez větších problému lze používat v rozšířených Java editorech jako Netbeans nebo Eclipse. Dále jsou pro kompilaci využity Ant skripty, takže program lze následně kompilovat i z příkazové řádky. Moduly jsou representovány separatními Java balíčky. Do programu se následně přidávají tak, že se jedná o metodou ZIP zabalené bytekódy. Výhodné je také to, že pro každy balíček lze automaticky vygenerovat JavaDoc dokumentace tříd a metod. S ohledem na výčet všech benefitů lze usuzovat, že použití JPF v implementovaném programu je celkem dobrou a rozumnou volbou.

5.2 Použití JPF v implementaci

Pro potřebu modularity vyvíjeného nástroje byl zvolen framework JPF. Aplikace se skládá z výchozího modulu `Core`, který se spustí po startu programu. Je inicializován subsystém pro načítání nastavení a

je zprovozněm systém pro registraci a zasílání událostí mezi moduly. Tento modul definuje rozšíření `Tool`, pomocí kterého lze připojit jakoukoliv komponentu, která definuje bod rozšíření daného typu, a to i za běhu programu. Pomocí tohoto rozšíření je inicializován daný načítaný zásuvný modul a jsou získány informace o něm. Následně je přidán jako záložka do okna programu. Obecně lze připojit jakýkoliv typ modulu, který definuje libovolné chování. Komunikace mezi moduly probíhá na základě zasílání událostí, pomocí kterých lze předávat cokoliv.

Návrh systému je tedy proveden tak, aby byla zajištěna maximální rozšiřitelnost. Dále je hlavní modul nezávislý na chování načítaných součástí a nabízí jim určitě využitelné služby. Chování modulů, jak již bylo zmíněno je plně v režii programátora.

5.3 Subsystém zasílání událostí

Jelikož moduly jsou mezi sebou plně nezávislé bylo nutno vyvinout přístup, který umožní rozumným způsobem předávat data a notifikace. Proto byl zvolen model založený na zasílaní událostí. Byly zváženy všechny možné alternativy nabízené jazykem Java. Následně bylo zvoleno nejpřijatelnější řešení.

Prvním přístupem, který byl uvažován bylo využití již stávajících tříd pro zasílání událostí. Ty fungují na principu takovém, že pro každou událost je vytvořen speciální objekt, do kterého je registrována obslužná metoda. Tento přístup je sice přijatelný, ale pokud by systém byl postaven na masivním zasílání události, je problém v tom, že počet objektů starajících se o obsluhu bude obrovský. Následně v tomto přístupu neexistuje možnost aby pro příjem jednoho typu události bylo registrováno více obslužných metod, například různých modulů.

Z důvodu značných nevýhod nativního řešení bylo přikročeno k řešení vlastnímu. Byl vytvořen subsystém umožňující registraci pro odběr událostí a také jejich zasílání. Programátor je schopen definovat vlastní třídu události a přiřadit typ řetězcem. Následně může každy modul využivající subsystém pro zasílání událostí registrovat obslužnou metodu pro daný typ události. To dále umožňuje odeslat událost více registrovaným příjemců najednou. Z pohledu implementace si třída pro práci s událostmi vede seznam jejich typů a obslužných metod. Umožní přidání nového typu události nebo registraci obslužné metody pro daný typ. Následně při požadavku o zaslání události jsou volány všechny registrované obslužné metody objektů pro daný typ zprávy. Je zde využito toho, že jazyk Java dovoluje volat metody a předávat jim parametry dokonce i v době běhu programu.

5.4 Modul pro tvorbu Modechart modelů

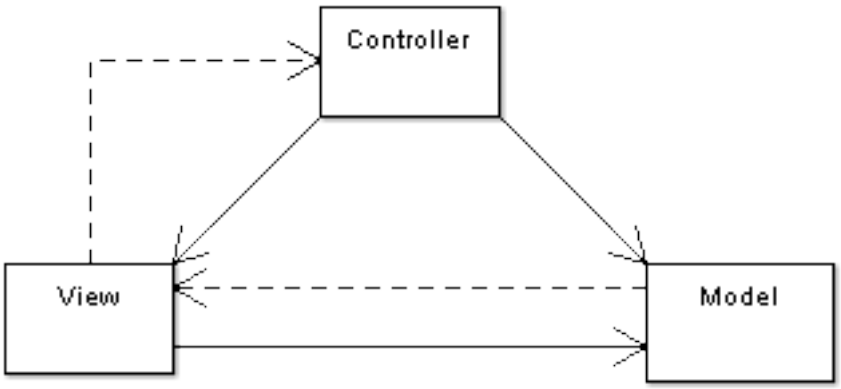
Prvním přídavným modulem, který je načten jádrem systému při jeho startu je subsystém, který umožňuje práci s Modechart spedifikací modelu. Je zde možno graficky navrhnout systém pomocí nakreslení hierarchické struktury modů a vedení přechodů mezi nimi. Uživatel může poté u každého druhu entity nastavit její parametry jako jsou jméno, typ, přechodové podmínky atd. Specifikaci lze následně uložit nebo načíst do/z XML souboru vybraného formátu.

Tato kapitola přiblíží základní koncepty a techniky využité v tomto modulu. Jsou nastíněny principy návrhového vzoru MVC, na jehož základě vznikla grafická a modelová část aplikace. Poté je popsán XML typ souboru a je přiblížen přístup k zpracování XML souborů různými technikami v jazyce Java. Jsou uvedeny výhody i nevýhody každé takovéto alternativy.

5.4.1 MVC pro práci s modelem

MVC je jedním z nejpoužívanějších architektonických vzorů, který je nejen využit v desktopových aplikacích, ale také v prostředí internetových aplikací. Hlavním cílem je rozdělení uživatelského rozhraní, jakožto prvků grafické uživatelské interakce, od datového modelu a aplikační logiky, která provádí požadované akce. To má za následek, že změnou jakékoliv takto dekomponované části není potřeba provádět zásadní zásahy do kódu samotného. Hlavními prvky tohoto vzoru jsou:

- Model (model), což je specifická representace informací zpracovávaných aplikací.
- Pohled (pohled), který převádí data modelu do podoby vhodné k interaktivní prezentaci uživateli.
- Controller (řadič), který reaguje na událostí (typicky pocházající od uživatele) a zajišťuje potřebné změny v modelu nebo pohledu.



Obrázek 5.3: Základní schéma návrhového vzoru MVC [8].

Na obrázku 5.3 je presentváno základní schéma propojení komponent tohoto vzoru. Klasické čáry popisují přímou asociaci a přerušované čáry asociaci nepřímou. Koncept MVC může být realizován různými zúůsoby, ale obecně platí následující princip:

1. Uživatel provede jistou operaci v uživatelském rozhraní (např. klik tlačítka myši).
2. Řadič obdrží informaci o provedení této akce.
3. Řadič v případě potřeby zaktualizuje model na základě provedené uživatelské akce (např. přidá novou entitu).
4. Model je jen jiným názvem pro doménovou vrstvu, která je schopná zpracovat změněná data.
5. Komponenta pohled používá zaktualizovaných dat modelu pro zobrazení pozměněných informací. Komponenta pohled získává data přímo z modelu, kde model nepotřebuje žádné informace o komponentě pohledu a je na ní absolutně nezávislý. Nicméně je možné použít návrhový vzor pozorovatel, který umožní modelu informovat o své změně jakékoliv registrované komponenty.
6. Uživatelské rozhraní čeká na další akci uživatele a tím se cyklus bodů opakuje.

5.4.2 Využití MVC v programu

Vyvinutý zásuvný modul má hlavně za úkol umožnit uživatelskou interakci s vytvářeným modelem. Hlavní okno se zkládá z komponenty, na kterou je umožněno kreslit a dále z menu, kde lze vybrat typ nástroje. Jsou definovány tyto tři typy:

- Výber entity, pričemž po značení myší jakékoliv entity dojde k zobrazení informací o ní.
- Kreslení modu, s nímž lze na základě kliku, tažení a uvolnění levého tlačítka myši vložit do obrázku nový mode.
- Kreslení přechodu, kde je umožněno klikem levého tlačítka myši na mode vložit počáteční bod, dále právym klikem tlačítka může uživatel umístit další body a následně levým klikem na mode vloží koncový bod přechodu.

Všechny uživatelské akce jsou směřovány do řadiče, který provede předepsané akce, kterými může být změna pohledu nebo modelu. Při označení entity modelu je možné pozměnit informace o ní. Každá entita má svou vlastní množinu atributů, které kterých může nabývat. Data jsou načítána z modelu a po jejich pozměnění jsou znovu uložena v modelu. Pro jednotlivé entity jsou definovány následující atribury. Entita typu mode definuje:

- Jedinečný název modu.
- Typ modu - buď sériový nebo paralelní.
- Poteční stav - udává jestli je mode počátečním stavem svého rodiče.
- Textový popis modu.

Následně pro entitu typu přechod jsou definovány atributy v závislosti na jeho typu, kde se může jednat o časovaný nebo spouštěný přechod. Proto je množina rozsáhlejší a obsahuje:

- Textový popis přechodu.
- Pokud se jedná o časovaný přechod tak:
  - Spodní časovou hranici.
  - Horní časovou mez.
- Pro spouštěný přechod jsou definovány podmínky v konjunktním tvaru, kde každá z nich může být:
  - Vstupní událostí tvaru $\rightarrow MODE1$.
  - Výstupní událostí definovánou jako $MODE1 \rightarrow$.
  - Událostí přechodovou formy $MODE1 \rightarrow MODE2$

Takto vypadá model zásuvného modulu z pohledu uživatele, nyní je potřeba zmínit implementační část řešení celého problému. Pohled je propojen s modelem a pozorovatelem. Pro tlačítka výběru nástroje je definován návrhový vzor pozorovatel, ke kterému je registrován řadič a na základě obdržených informací je schopen volit potřebné kroky. Všechny události jako kliky či tažení myši jsou směřovány také do řadiče, který dokáže pozměňovat grafickou podobu komponent, jako

například jejich barvu či pozici. Z modelu zase na druhou stranu jsou načítány hodnoty atributů, které mohou být změněny a po potvrzení tohoto kroku zpětně uloženy.

Jednotlivé entity v pozorovateli vycházejí z bázového rozhraní `IMGUIDrawable`, které definuje metody pro zjištění aktivity, typu objektu a průniku bodu s daným objektem. Dále je zde definována metoda `execute`, jejíž implementací každý poděděný objekt definuje svou grafickou podobu. Vprogramu jsou definovány tři typy objektů grafických komponent:

- `MGUIDrawableMode`, representující čtverec modu.
- `MGUIDrawableTrans`, který popisuje přechod mezi mody.
- `MGUIDrawableTransKnot`, jenž je díky použití návrhového vzoru *composite* součástí třídy přechodu a definuje jednotlivé jeho dílčí body.

Všechny třídy implementují metodu pro své vykreslení a dále i pro zjištění aktivity. Další metoda, která je nesmírně důležitá, je zjištění průniku z bodem a na jejim základě je nalezen po kliku myši označený objekt. Pro vykreslování jednotlivých entit je využito návrhového vzoru *command*. Ten uchovává ve vektoru reference na všechny vykreslované objekty a následně, pokud je potřeba je schopen postupně zavolat metodu `execute` nad všemi grafickými entitami a najednou je všechny vykreslit do obrázku. Pro vizualizaci je využito plátno, které je implementováno třídou `MGUIJPanel` poděděnou ze standardní Java třídy `JPanel`. V této třídě je přetěžena metoda `paint` a je zde pro vykreslování prvků použita již dříve zmíněná třída `MGUIDrawableCmd`. Je také podporována technologii double bufdering, takže ne vždy jsou vykreslovány všechny prvky. Proto například při kreslení nového přechodu je uložena kopie obrázku na něj se jen vykresluje grafický objekt postupně jak mění souřadnice. Pro účely možnosti změny informací o entitě je definována třída `MGUIPropertiesTab`, která je schopná do předem definovaných políček načíst informace o atributech entity. Následně po potvrzení změn jsou data zpětně uložena do modelu.

Všechny události vyvstalé v pohledu jsou směřovány do řadiče. Zde se nachází většina logiky programu, jelikož na základě vyhodnocení událostí jsou voleny konkrétní kroky. Nejsložitější součástí je model, v němž jsou uloženy všechny zásadní a podstatné informace. Model specifikuje dvě základní třídy `MDModeManager` a `MDTransManager`. První z nich obsahuje stromovou strukturu modů a umožňuje provádět operace nad nimi. Další třída definuje vektor přechodů a nabízí operace, které lze nad nimi provádět. Pro jednotlivé entity jsou vytvořeny objekty:

- `MDMode`, representující mode včetně dat o něm.
- `MDTrans`, který popisuje přechod mezi mody a obsahuje potřebné informace o něm.

Pro každou entity jsou definovány třídy poděděné z `IEntityProperty`, které definují množiny atributů entity a jejich hodnoty. Jmenovitě se jedná o MDModeProperty obsahující jméno, typ, popis a hodnotu jestli se jedná o počáteční stav. Pro přechod je definována třída `MDTransProperty`, která obsahuje informace o typu přechodu a následně buď časové nebo spouštěcí podmínky. Hodnoty takovýchto objektů poté mohou být měněny uživatelem v grafickém rozhraní pohledu.

5.4.3 XML formát uložení modelu

Tento modul umožňuje tvorbu modelů realtime systémů a je nutno zhodnotit do jakého formátu souboru bude možno data ukládat. Je nutno brát ohled na to, aby použitý přístup byl jednoduchý na implmentaci a co nejlépe využil již existující nástroje nabízené jazykem Java. Model sám o sobě je velice komplexní a skládá se z mnoha částí. Obecně lze ale problém specifikovat tak, že se pokoušíme uložit data o uzlech a hranách mezi nimi a reflektovat hierarchickou strukturu. Pokud ještě

uvažujeme použití návrhového vzoru MVC, skládá je model nejen z těchto dat ale také z informací o grafické podobě prvků. Proto můžeme rozdelit ukládaná data podle jejich druhu a typu entit, které popisují. Lze soudit, že musíme ukládat tyto informace:

- Uzly modelu a jejich hierarchické uložení.
- Data přechodů mezi uzly.
- Data modelu jako např. jméno, typ či přechodové podmínky.
- Grafické aspekty jako pozice bodů obdelníku Modechart modu nebo body přechodů.

Je nutno poznamenat, že každý typ entity má některé atributy stejné a část odlišnou. Proto pro tento účel byl zvolen formát souboru XML. Tento dovoluje specifikovat hierarchickou strukturu, což bude oceněno při ukládání stromu Modechart modů. A také umožní definovat různé typy položek pro každou ukládanou entitu.

Použití XML v jazyce Java

Existují dva rozdílné způsoby analýzy XML v jazyce Java: SAX a DOM. SAX je zkratkou pro Simple API for XML a umožňuje interpretaci značek XML v datech tak, jak na ně parser postupně naráží. Jinými slovy pokud parser na XML značku, zavolá obslužnou metodu, která se postará o zpracování. Je na programátorovi, jaké následné akce provede. DOM je následně zkratnou pro Document Object Model. Nejedná se striktně o API, nýbrž o popis organizace elementů XML dokumentu. Pokud je dokument analyzovám DOM parserem, je vytvořen objekt typu *Document*, který obsahuje všechna data, která byla definována v XML dokumentu.

Každý z jmenovaných přístupů má své výhody i nevýhody a záleží jen na situaci a účelu využití, který přístup zvolit. Například pokud programátor, bude chtít zpracovácat velké soubory řádu desítek megabytů, pravděpodobně zvolí SAX, jelikož přístup pomocí DOM musí načíst celý dokument do paměti, což samozřejmě znamená značné zpomalení.

SAX používá model interpretace na základě zasílání událostí. SAX parser postupně čte XML dokument a pokud narazí na rozumná data, zavolá obslužnou metodu. Obslužná metoda vychází z již definovaných prototypů a zastřešuje užitečný kód, který na základě dat elementu provádí zamýšlené akce. SAX informuje o každém startující a ukončující značce či znaku dat v rámci elementu. Také hlásí chyb, pokud na nějaké při čtení dokumentu narazí. Může se například jednat o neplatné znaky nebo neukončení značky. Tento typ přístupu je nejužitečnější, když je potřeba analyzovat velice velký XML soubor a je potřeba získat jen část užitečných dat. Pokud je potřeba prohledat XML strom z důvodu nalezení určitých dat, tento přístup je velice rychlý.

DOM parser musí načíst celý soubor před tím, než lze získat užitečná data. Vytvoří hierarchickou interní representaci XML obsahu s použitím Java tříd. XML má strukturu stromu, kde hlavní značka dokumentu je kořenem a vnořené značky jsou potomky. DOM pracuje také na principu stromu a plně reflektuje rierarchii zdrojového souboru. Po načtení souboru je vytvořen objekt *Document*, který obsahuje objekty vycházející z rozhraní *Node*, jenž representují jednotlivé potomky. Z rozhraní *Node* jsou vytvořeny různé další třídy. Nejčastější implementací je třída *Element*, která popisuje značku nebo dvojici značek XML dokumentu. Dalšími třídami jsou například *Comment*, *Text*, *CDATASection*, *Character* a mnoho dalších. Třída *Element* může obsahovat synovské uzly představující značky a data obsažená mezi počátkem a koncem značky.

Implemtace podpory XML v programu

Pro účel programu byl zvolen XML parser, jelikož na rozdíl od SAX přístupu umožňuje i flexibilní vytváření nových dokumentů. Proto v následné implementaci je čtení i zápis XML realizovaán pomocí DOM. Vytvoření dokumentu z Modechart modelu je velice jednoduché, jelikož se jedná o opačný postup ke čtení souboru a DOM nabízí možnost postupné konstrukce stromu XML.

Na základě analýzy byla zvolena vhodná struktura dokumentu, který je rozdělen do dvou základních částí: *modes* a *transitions*. Tímto dochází k rozdělení dokumentu na dvě části s ohledem na uvažované entity. Dále v rámci každé položky jsou ve značce *properties* uložena data MVC a to jmenovitě *model* a *view*. Každý typ polžky ale definuje různá obsažená data. Následně z důvodu reflektování hierarchické struktury je v entitě *mode* definována začka *children*, která obsahuje přímé potomky.

Typ entity mode obsahuje tyto položky:

- Souřadnice obdelníku modechart v jeho grafické representaci.
- Data modelu jako:
  - Jméno modu.
  - Informace o tom, jestli se jedná o počáteční mode.
  - Typ modu, buď paralelní nebo sériový.
  - Textový popis.

V rámci definice entity typu přechod lze také definovat následující:

- Souřadnice všech bodů přechodu.
- Data modelu jako:
  - Výchozí a cílový mode.
  - Textový popis entity.
  - Typ přechodové události. Pokud se jedná o časovanou událost tak spodní a horní časovou hranici. Jestliže jde o spouštěný přechod je obsažen výčet všech konjunktních událostí.

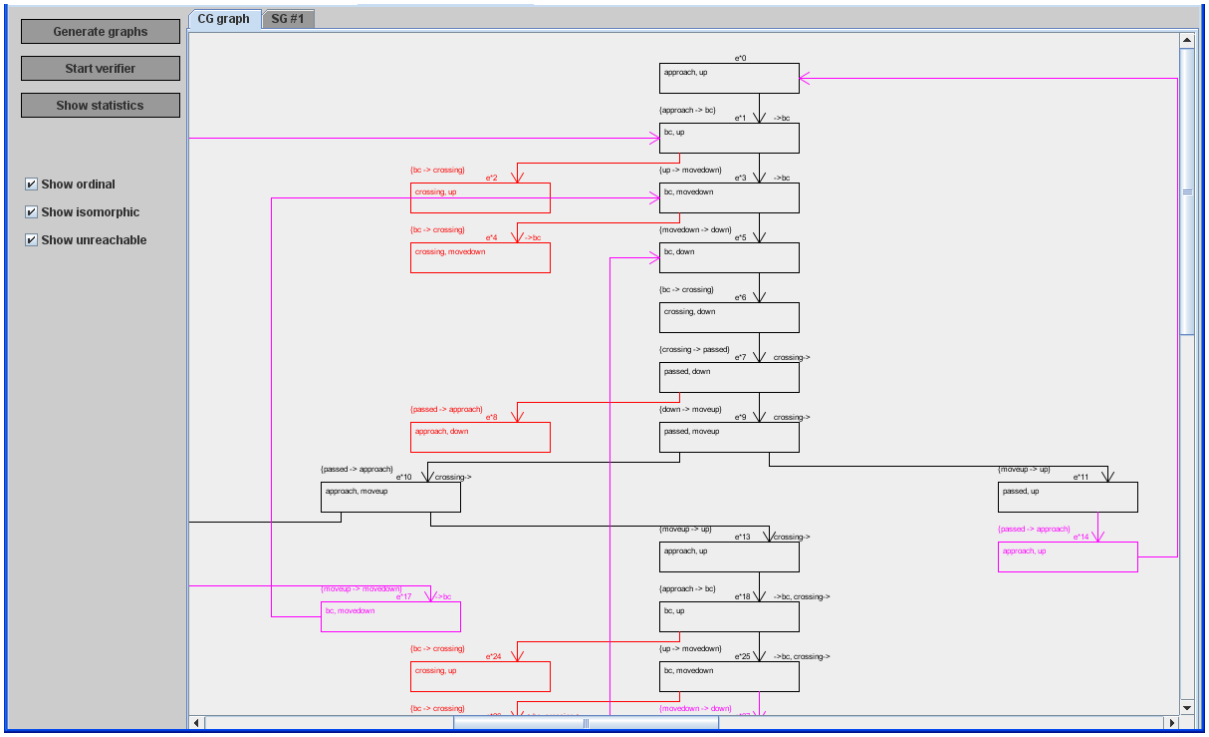
Program plně využívá možností jazyku Java pro práci s XML a také je čerpánu ze všech výhod tohoto přístupu k popisu dokumentů. Subsystém pro práci s XML byl úspěšně testován a to i na rozsáhlejších modelech obsahujících mnoho entit. Jak bylo zjištěno, výsledné modely zabírají místo na disku v rozmezí cca. 10-20Kb, což je velice přijatelná velikost souboru.

5.5 Modul pro analýzu a verifikaci

V této kapitole je přiblížena struktura a funkcionality modulu nabízejícího analýzu a verifikaci modelu specifikovaného formalismem Modechart. Jsou popsány funkcionality nabízené grafickým uživatelským rozhraním. Následně jsou přiblíženy vizuální vlastnosti separačního a výpočetního grafu. Nástíněna je také jejich interní struktura a způsob konstrukce. Dále je popsán způsob implementace algortimu pro převod Modechart modelu do podoby zmíněných grafů. Je ukázano, jakým způsobem lze provádět verifikaci formulí RTL logiky oproti vytvořenému modelu. Dalším obsahem kapitoly je popis přístupu ke generování formulí RTL logiky ze získaného modelu pro účel následného zpracování dalšími nástroji. Modul také umožňuje výpis statistik, proto i popis tohoto je zahrnut do následujícího textu.

5.5.1 Grafické aspekty a funkcionality

Grafické rozhraní se skládá ze dvou hlavních části jimiž jsou visualizace grafů a ovládací prvky. Jsou vhodně graficky zobrazeny separační a výpočetní graf a lze mezi nimi libovolně přepínat. Na levé straně je nacházejí ovládací prvky. Stisknutím tlačítka je možno vygenerovat grafy ze specifikace otevřené v modulu pro návrh Modechart. Další tlačítko spouží pro spuštění verifikačního nástroje, jimž lze ověžit formule oproti grafům systému. Posledním tlačítkem je možno zobrazit přehledné statistiky. Existují také tři zaškrkávací políčka, které umožňují specifikovat jaké typy entit se v grafech mohou zobrazovat. Jejich vhodným nastavením se grafy mohou stávat přehlednější. Jak vypadá grafické rozhraní modulu je ukázáno na obrázku 5.4.



Obrázek 5.4: Hlavní okno modulu pro analýzu a verifikaci.

5.5.2 Tvorba CG grafu

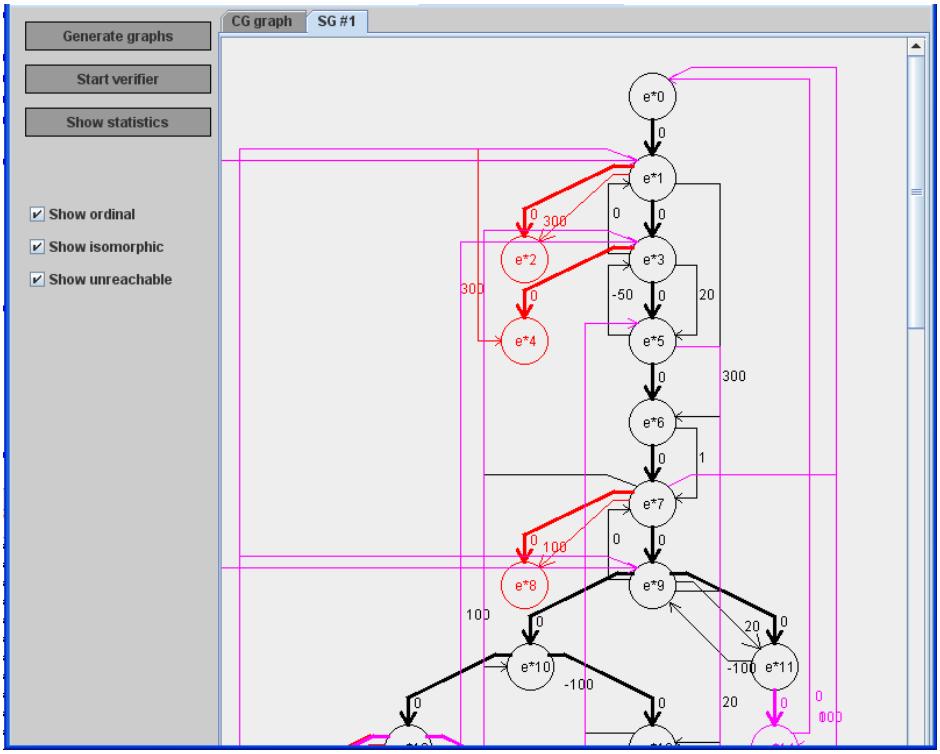
Výpočení graf plně vystihuje všechny možné výpočetní posloupnosti systému. Tyrkysovou barvou jsou zvýrazněny isomorfní uzly a hrany. Červená barva za to specifikuje uzly nedosažitelné. K popisu uzlu patři čtyři základní atributy:

- Globální konfigurace.
- Množina událostí uzlu.
- Přechodová událost, která způsobila vstup do uzlu.
- Čislo přechodové události separačního grafu.

Na základě tohoto grafu lze zkoumat chování systému a vizuálně ověřit jeho funkčnost. Bohužel u větších modelů jsou grafy rozsáhlé, proto pro ověření funkčnosti nezbývá než sáhnout po verifikačním nástroji. Jak již bylo zmíněno tyrkysová barva, zde vystihuje isomorfní uzly, čili takoví, které byly sloučeny s nějakým jiným uzlem z důvodu toho, že podstromy těchto dvou zlů jsou isomorfní. Nedosažitelné uzly specifikují stavy, do kterých se systém mohl dostat, ale porušil by časová integritní omezení. Jedná se tedy o možné následníky, kteří ale nebyli vybrání za skutečné. Tento graf je postupně budován již dříve popsaným algoritmem převodu a je ukázán na obrázku 5.4.

5.5.3 Separační graf

Separační graf je velice důležitou komponentou, která popisuje časová omezení uzlů výpočetního grafu. Existují zde dopředné hrany specifikující minimální časovou vzdálenost dvou událostí. Také jsou popsány hrany zpětné, které definují maximální vzdálenost událostí. Na základě analýzy vzdáleností v grafu funguje jak výpočet skutečných následníků, tak je použita pro následnou verifikaci. Vzhled separačního grafu je na obrázku 5.5. Barvy uzlů zde mají stejný význam jako u výpočetního grafu a lze volně přepínat mezi typy, které se budou zobrazovat.



Obrázek 5.5: Separační graf výpočetního modelu.

Pro vý počet separační ch vzdálenosti uzlů byla vyvinuta speciální procedura. Bohužel nexistuje rozumný algoritmus pro výpočet maximální vzdálenosti dvou zlů. Problémem je to, že daný graf obsahuje cykly, které maji ke všemu negativní váhu. Proto bylo přikročeno k prohledávání stavového prostoru do šířky pomocí algortimu limited BFS. Tento algortimus vykazoval dostačující rychlost a paměťovou náročnost.

5.5.4 Implementace algoritmu konstrukce grafů

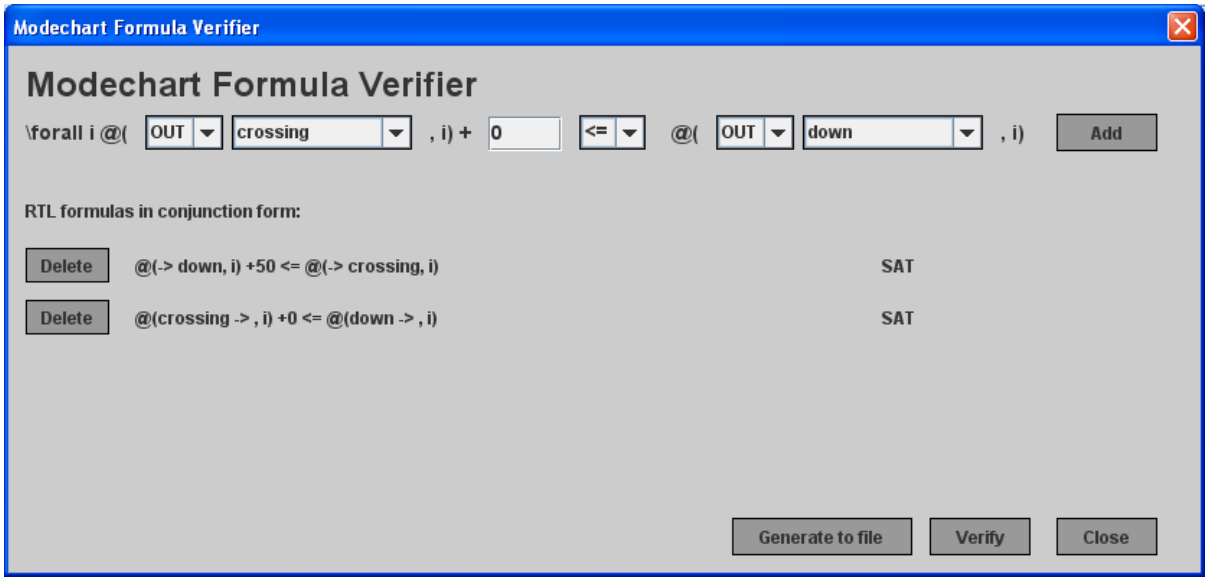
Pro generování je nejprve nutno získat otevřený model z modulu pro návrh Modechart. K tomuto účelu je využit subsystém zasílání událostí, pomocí kterého je předána reference na model. Prvky odelu jsou následně vhodnou funkcí převedeny do pro analýzu efektivnější representace. Tento přístup značně zvyšuje rychlost všech operací prováděných nad prvky modelu, jelikož všechny vytvořené objekty mají velice jednoduchou strukturu a nesou jen potřebné informace.

Byl implementován již dříve popsaný algortimus pro převod modelu do podoby výpočetního a separačního grafu. Bylo nutno vytvořit metodu `Reach`, která zajistí zjištění nového globálního modu po provedení zvoleného přechodu. Dále bylo vhodně zajištěno dědění událostí. Pro výpočet množiny možných následníků byla vyvinuta funkce, která dokáže zjistit množinu přechodů, které lze volit z daného globálního modu. Poté po použití separačního grafu a spočtení vzdálenosti mezi možnými následníky je vytvořena množina aktuálních následníků.

Algoritmus byl testován na Modechart specifikaci ukázkových příkladů a vykazoval dostatečnou rychlost převodu a výsledný model se shodoval s očekávaným. Jak již bylo zmíněno, tak některé úlohy generují celkem veliký stavový prostor grafů. Poté je problém takovýto výsledek vhodně visualizovat.

5.5.5 Verifikace vlastností modelu

Verifikační program umožňuje generovat výpočetní model z Modechart popisu systému. Tyto grafy, jak již bylo několikrát zmíněno plně vystihují výpočetní trajektorii popisovaného systému. Tedy dokáží popsat všechny možné běhy systému, kterých je sice nekonečně ale spočetně mnoho. Je zřejmé, že popis pomocí grafů v sobě nese značnou vypovídací schopnost. Proto byly implementovány algortimy a rozhodovací procedury, které umožňují ověřit jisté tvrzení oproti výpočetnímu modelu. Získané informace nám mohou říci, jestli byl systém dobře navržen. Pokud tomu tak je, je možné zkoumat časové závislosti jednotlivých přechodových událostí systému.



Obrázek 5.6: Okno verifikátoru s ověřením formulí vůči modelu železničního přejezdu.

Okno obsahující prvky verifikačního nástroje lze otevřít po vygenerování grafů systému z hlavní

nabídky modulu pro verifikaci. Jak vypadá je ilustrováno na obrázku 5.6. Ověřované formule jsou v již dříve zavedeném tvaru $@(E_1, i) \pm I \leq (E_2, i)$, kde E_1 a E_2 jsou událostmi, i je výskytovým indexem a I je celočíselná konstanta vystihující časovou separaci. Pro události s ohledem na popis Modechart specifikace definujeme událost vstupu a výstupu z daného modu. Symbolicky označováno šipkou před resp. za názvem modu. Tento typ lze při zadávání ověřované formule vybrat ze seznamu stejně jako název modu, se kterým je daný typ události spjat. Poté je možno zadat číslo popisující časovou separaci těchto dvou událostí. Po kliku na tlačítko je tato formule přidána do seznamu ověřovaných formulí, které mohou být celkově až čtyři. Jakoukoliv formuli lze následně podle uvážení i smazat. Všechny formule, které jsou zadány do seznamu jsou chápány jako konjunkce obecně kvantifikovaných predikátů. Pro příklad na obrázku bude ověřena pravdivost tvrzení: $\forall i(@(\rightarrow down, i) + 50 \leq @(\rightarrow crossing, i) \wedge @(crossing \rightarrow, i) + 0 \leq @(down \rightarrow, i))$. Pokud je tvrzení takto kvantifikováno, bude ověřena jeho pravdivost pro všechny související koncové body obsahující danou událost. Tedy pokud bude formule splněna, bude striktně platit pro celý výpočetní model. Ověření formulí se provede klikem na tlačítko ve spodní části okna. Následně bude u každé formule zobrazeno její splnitelnost, resp. nesplnitelnost. Jak vyplývá z definice konjunkce, aby celý výray byl pravdivý, musí všechny jeho členy definovat pravdivou hodnotu. Pomocí dalšího tlačítka lze generovat logické RTL formule popisující systém do textového souboru. Tento princip je popsán v následující podkapitole.

Z pohledu implementace je pro representaci formule vytvořena třída *MDVerifFormula*, která obsahuje informace o jedné z maximálně čtyř ověřovaných formulí. Z obecného tvaru je formule převedena na tvar ve formě:

- $I \leq e_2 - e_1$ nebo
- $e_1 - e_2 \leq I$

V tomto vyjádření e_1 a e_2 specifikují výskytové funkce jednotlivých událostí a I je jejich časovou separací. Na formuli prvního tvaru pro ověření její pravdivosti je aplikována již dříve popsaná rozhodovací procedura *checkmindistance*. Splnitelnost druhého typu je potvrzena nebo vyvrácena procedurou *checkmaxdistance*. Pro jednotlivé události jsou nalezeny všechny související koncové body. Jelikož je formule kvantifikována obecně, je nutno dokázat splnitelnost pro každou dvojici koncových bodů. V opačném případě pro formuli s existenčním kvantifikátorem by stačilo aby vztah byl splnitelný pro jednu dvojici souvisejících koncových bodů.

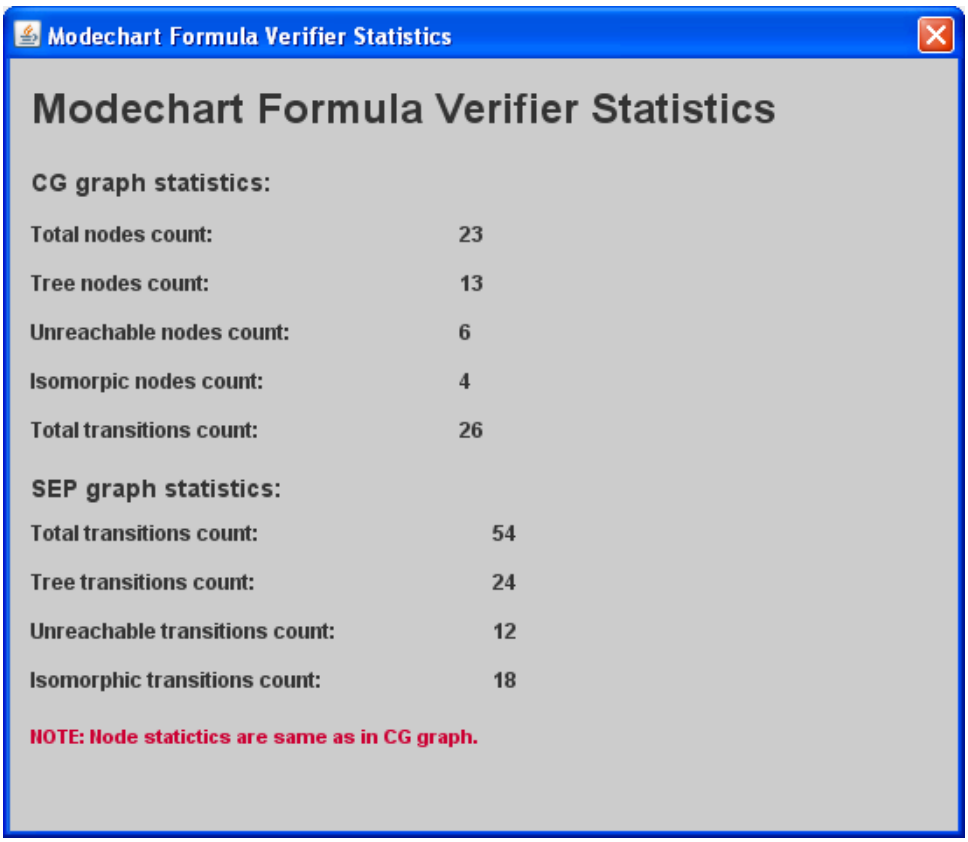
Ověření vzdáleností je provedeno nad již nagenerovaným separačním grafem popisovaného modelu. Pro potřebu zjištění maximální vzdáleností dvou bodů separačního grafu je použita pozměněná procedura, která je využita pro hledání skutečných následníků uzlu. Minimální vzdálenost je počítána obdobným způsobem. Je potřeba ale podotknout, že jelikož graf obsahuje negativní cykly, je nutno je detekovat, jelikož vedením několika opakujících se průchodů takovýmto cyklem opakovaně snižuje hodnotu minimální vzdálenosti. Proto není dovoleno procházet cyklem a tovéto vypočtené vzdálenosti jsou ignorovány. Dále jsou pro daný případ ignorovány stromové hrany s váhou 0, protože jejich vedením by byla minimální vzdálenost mezi jakýmikoliv dvěmi uzly rovna nule. Pro oba případy je použit dříve popsaný algoritmus limited BFS.

5.5.6 Generování formulí RTL logiky

Separační graf jako takový pomocí uzlů vystihuje přechodové události a hrany definují časové saparace mezi nimi. Byly proto zkoumány způsoby, jak z daného grafu generovat formule RTL logiky a bylo přihlédnuto i k následné redukci jejich stavového prostoru.

Bylo zjištěno, že pokud verifikujeme systém oproti nějakému předem známému bezpečnostnímu pravidlu, jsme schopni omezit zkoumaný interval separačního grafu a tím i snížit počet generovaných formulí. Pokud do Modechart verifikátoru zadáme formule pro ověření, prakticky definujeme omezení na vzdálenosti mezi koncovými body. Dva korespondující koncové body vždy tvoří interval. Proto musíme v separačním grafu najít takový interval aby pokrýval všechny intervaly mezi korespondujícími koncovými body. Dále musíme takovýto vzniklý interval rozšířit na takovou velikost aby pokrýval i všechny uzly, které mají časované přechody vedouci dovnitř intervalu. Jelikož pokud se tomu tak nestane, nebudou vhodně reflektována časová omezení v rámci intervalu. Tímto přístupem lze dosáhnout značné redukce počtu formulí.

Výpočetní model lze převést pomocí tlačítka ve verifikátoru a následně generovaná data jsou uložena do souboru. Jsou vytvořeny formule omezené RTL logiky specifikující separaci událostí. Jména událostí jsou totožná s označením uzlů separačního grafu, což zavádí korespondenci mezi nimi a umožňuje následnou vizuální kontrolu.



Obrázek 5.7: Ukázka statistik modelu.

5.5.7 Analýza modelu a statistiky

V předešlých podkapitolách byl popsán CG a také SEP graf. Bylo ukázáno z jakých hlavních entit se tyto grafy skládají a jaké mohou mít atributy. Na obrázcích bylo možno pozorovat grafickou podobu těchto struktur včetně barevných rozlišení entit podle nabývaných atributů. Sice byla dobře patrná struktura, ale postrádalo se vyjádření pomocí kvantitativních činitelů. Proto byla do vyvíjeného nástroje přidána součast, která umožňuje zobrazit statistiky CG a SEP, resp. celého výpočetního

modelu. Zde si může uživatel programu na základě zobrazených hodnot udělat představu o komplexnosti a náročnosti vytvořeného modelu a následně z těchto informací odvodit např. časovou či paměťovou náročnost následné verifikace. Na obrázku 5.7 je ukázka výstupu tatovýchto statistik pro celkem jednoduchý model.

Je patrné, že statistiky jsou separátně generovány pro každy graf. Jelikož některé atributy, jak je i poznamenáno vespod obrázku, jsou pro oba grafy stejné, nebyly zavedeny duplicitní položky. Pro výpočetní graf je v první řadě vypsán celkový počet uzlů a hran, kde jsou započítány všechny jejich druhy. Poté je jasně viditelné, že je udán počet stromových uzlů, čili uzlů, které jsou dosažitelné a nejsou isomorfní. Následně jsou uvedena data pro počet nedosažitelných a isomorfních uzlů. Jejich počet silně závisí na povaze zkoumaného modelu a to i na úrovni nedeterminismu, tak vhodné volbě spouštěcích podmínek a časování přechodů. Jelikož korespondence mezi uzly výpočetního a separačního grafu je jedna ku jedné, nejsou pro druhý typ grafu uvedeny statistiky uzlů. Jelikož ale pro následnou verifikaci je nesmírně důležitý počet hran SEP grafu, je zde tato informace uvedena. Podobně jako u popisu uzlů, jsou přechody na základě atributů spojovaných entit rozděleny na stromové, nedosažitelné a isomorfní.

Kapitola 6

Specifikace jednoduchých úloh a zhodnocení funkčnosti

Z důvodu nutnosti ověření funkčnosti vyvíjeného systému bylo vybráno několik úloh. Celkem se jedná o tři slovně popsané příklady specifikující systém a jeho časová omezení. Ke každé definici je dospecifikováno bezpečnostní tvrzení, jehož pravdivost budeme mít za úkol ověřit. Následně byl vybrán příklad železničního přejezdu, který byl převeden do podoby modelu formalismu Modechart. Nad tímto bylo demonstrováno použití dvou rozdílných technik verifikace.

6.1 Neformální definice úloh

6.1.1 Železniční přejezd

Přejezd obsahuje pouze jedinou kolej, v určitý čas tedy může projíždět pouze jediný vlak. Systém je složen z komponent vlaku, vlakového senzoru, ovladače závor a samotných závor. Úkolem ovladače závor je zajistit, aby v době průjezdu vlaku železničním přejezdem se na křížení cesty a kolejí nevyskytovalo žádné auto. Lze předpokládat, že tento úkol je vždy splněn, pokud jsou závory v době průjezdu vlaku sklopeny.

Specifikace systému:

- Když se vlak přiblíží k vlakovému senzoru a je zachycen tímto senzorem, je zaslán signál ovladači závor, který začne pomalu sklápět závory před železničním přejezdem.
  - Vlak začne projíždět přejezdem nejrychleji 300 sekund po detekci senzorem.
  - Sklopení závory trvá alespoň 20 sekund ale ne více než 50 sekund.
  - Senzor dokáže detekovat opuštění přejezdu vlakem a poté je potřeba minimálně 20 sekund a ne více jak 100 sekund na zvednutí závory.
  - Jelikož jsou příjezdy vlaků dobře naplánovány, po upuštění přejezdu vlakem, může další vlak začít přijíždět až za 100 sekund.

Bezpečnostní tvrzení:

- Z důvodu bezpečnosti musí být závory dole minimálně 50 sekund před okamžikem, kdy vlak začne vjíždět na přejezd. Dále ke zvedání závor dojde až když vlak opustí přejezd.

6.1.2 Raketový systém stíhačky

Stíhačka je vybavena raketami na boj na dálku. Po zaměření rakety je nutno ji vypustit. Je potřeba zajistit, aby došlo ke včasnému otevření zbraňové šachty a aby raketa zažehla motor v bezpečné vzdálenosti od letadla. Také je nesmírně důležité, aby se raketa stihla před zásahem cíle odjistit.

Specifikace systému:

- Při aktivaci systému pro zaměření cíle dojde k zahájení otevírání zbraňové šachty.
- Otevírání zbraňové šachty bude dokončeno do 4 sekund od aktivace systému pro zaměřování.
- Uplné otevření zbraňové šachty trvá alespoň 2 sekundy.
- Po otevření šachty lze kdykoliv vypustit raketu pokud je zaměřen cíl, ta se nejprve uvolní ze zbraňové šachty, což trvá maximálně 1 sekundu, poté začne padat pod letadlo.
- Po uvolnění rakety bude zbraňová šachta jestě 3 sekundy otevřena a poté se začne zavírat.
- Uzavření je stejně časově náročné jako otevírání, tzn. trvá minimálně 2 sekundy a maximálně 4 sekundy.
- Po uplynutí 1 sekundy od uvolňení rakety dojde k aktivaci polohového čidla, které vyhodnocuje pozici rakety vzhledem k letadlu.
- Čidlo vždy maximálně do 2 sekund vyhodnotí, že vzdálenost od letadla je bezpečná pro zažehnutí raketového pohonu (čas se může lišit podle rychlosti pádu rakety a směru pohybu stíhačky, ale při jakémkoliv pohybu a manévrech se stíhačka do 2 sekund dostatečně vzdálí od rakety).
- Podle pohybu rakety a stíhačky poté dojde do 3 sekund od zažehnutí raketového pohonu k odjištění rakety.
- Odjištění rakety je ale vždy provedeno alespoň po 2 sekundách.

Bezpečnostní tvrzení:

- Pokud bude zaměřování cíle dokončeno do 5 sekund od okamžiku započetí zaměřování a pilot do 2 sekund po zaměření vypustí raketu, je zaručeno, že do 15 sekund bude k nepřátelskému letadlu mířit odjištěná raketa.

6.1.3 Kulomet stíhačky

Stíhačka je vybavena výkonným kulometem pro boj z blízka, ten se ovšem při střílení zahřívá a pokud zahřátí překročí únosnou, mez může dojít k jeho poškození a znefunkčnění. Této situaci je třeba zabránit, například tak, že kulomet bude obsahovat čidlo detekující jeho teplotu a při větším zahřátí zastaví jeho činnost. Ovšem je nutné také zajistit, aby po opětovném poklesu teploty kulometu bylo možné znova střílet, což může opět zajistit přítomné čidlo.

Specifikace systému:

- Kulomet se nezačne zahřívat dříve než po 20 sekundách střílení.
- Teplota kulometu překročí únosnou mez do 10 sekund od počátku zahřívání.

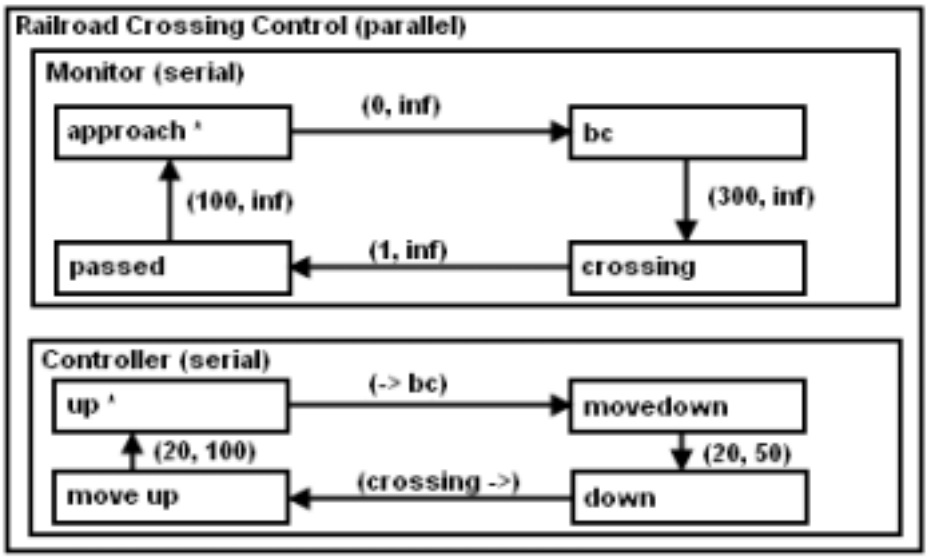
- Úplné zahřátí kulometu trvá vždy ale nejméně 7 sekund.
- Ochlazování kulometu začne probíhat ihned po jeho vypnutí.
- Zchladnutí kulometu na přijatelnou teplotu trvá podle klimatických podmínek alespoň 3 sekundy, ale maximálně 5 sekund.
- Ihned po detekci zchlazení kulometu dojde k opětovnému povolení střelby.

Bezpečnostní tvrzení:

- Pokud čidlo detekuje zahřátí kulometu do 5 sekund od počátku zahřívání a pokles teploty kulometu do 3 sekund od ustálení přijatelné teploty je zaručeno, že nikdy nedojde k přehřátí kulometu a k opětovnému povolení střelby kulometu po zchlazení dojde do 10 sekund od vypnutí kulometu.

6.2 Přepis do Modechart a ověření funkčnosti

Jelikož úloha železničního přejezdu je nejjednodušší, bude na ní demonstrován přístup k vytvoření Modechart modelu, následné verifikaci a generování RTL formulí spojené s ověřením jejich pravdivosti oproti bezpečnostnímu tvrzení. Slovní úloha byla analyzována a byl definován Modechart model. Dále bylo přepsáno bezpečnostní tvrzení do RTL logiky ve tvaru $\forall i @(\rightarrow down, i) + 50 \leq @(\rightarrow crossing, i) \wedge @(crossing \rightarrow, i) \leq @(down \rightarrow, i)$. Vytvořený Modechart model je na obrázku 6.1. Pro ilustraci lze odkázat na výpočetní graf 5.4 a obrázek separačního grafu dané úlohy 5.5.



Obrázek 6.1: Modechart model železniního přejezdu.

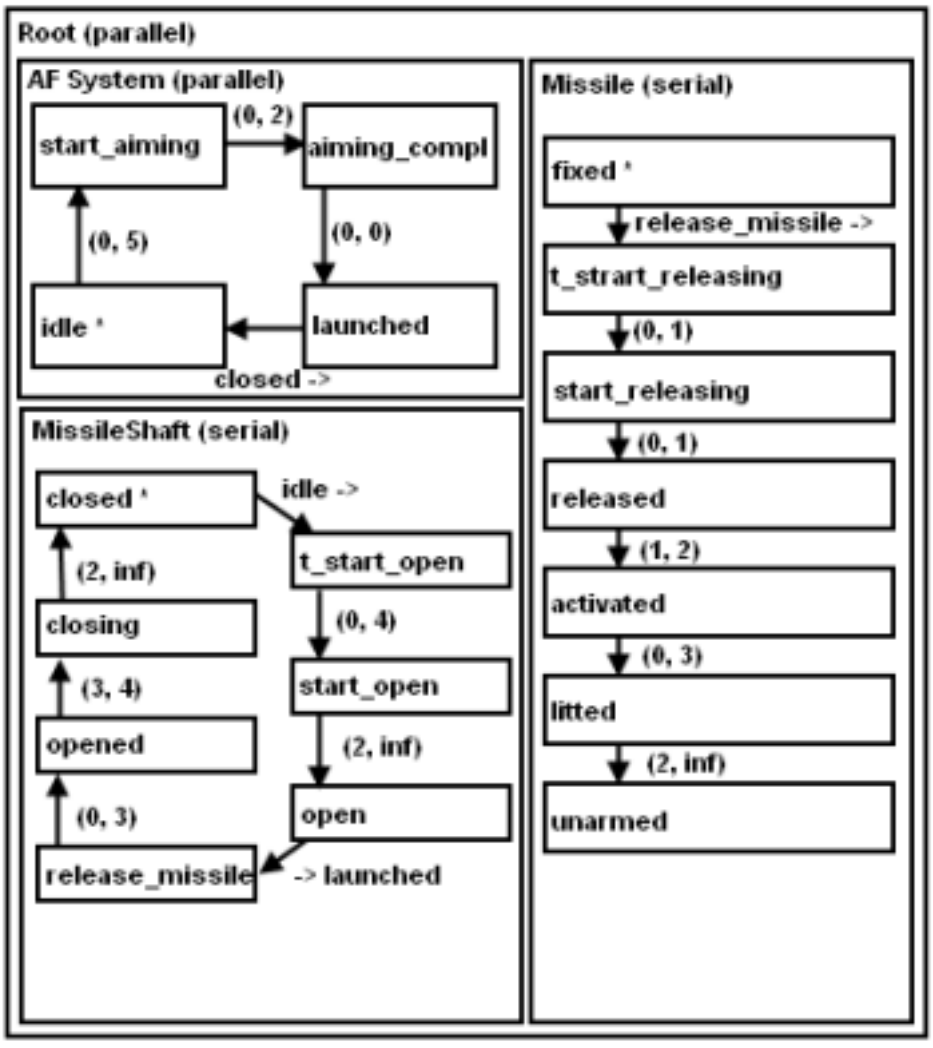
Popsaný model byl úspěšně verifikován oproti bezpečnostnímu tvrzení, jak je patrné na obrázku 5.6. Následně byla ještě ověřena funkčnost generování formulí RTL logiky. Nad těmito byla spuštěna procedura pro redukci jejich počtu. Výsledné formule jsou uvedeny níže.

```
V i(@(%E1, i) + 300 =< @(%E6, i) and
@(%E1, i) + 0 =< @(%E3, i) and
```

```
@(%E3, i) + 0 =< @(%E1, i) and
@(%E3, i) + 20 =< @(%E5, i) and
@(%E5, i) - 50 =< @(%E3, i) and
@(%E5, i) + 0 =< @(%E6, i) and
@(%E6, i) + 1 =< @(%E7, i) and
@(%E7, i) + 0 =< @(%E9, i) and
@(%E9, i) + 0 =< @(%E7, i))
```

Nad tímto matematickým popisem byla spuštěna rozhodovací procedura vyvíjená Bc. Janem Fiedorem. Na základě jejího výsledku bylo shledáno, že dané bezpečnostní tvrzení platí. Je nutno poznamenat, že tyto formule přímo vycházejí ze separačního grafu a z důvodu zachování korespondence s ním byly ponechány názvy přechodových událostí.

Další dva příklady byly také převedeny do podoby modechart. Jak vypadá příklad modelu raketového systému stíhačky je na obrázku 6.2



Obrázek 6.2: Modechart model raketového systému stíhačky.

Všechny tyto příklady jsou součástí adresáře vyvíjeného programu a lze si je bez problémů spus-

tit. Je poté možné zobrazit jejich výpočetní a separační grafy. Je nutno ale poznamenat, že tyto jsou celkem rozsáhlé. Dále je možnost ověření funkčnosti pomocí spuštění rozhodovací procedury. Bylo shledáno, že převod do modelu formalismu Modechart byl úspěšný a dále pro úlohu železničního přejezdu byly shledány pozitivní výsledky verifikace pomocí dvou rozdílných přístupů.

Kapitola 7

Závěr

V první části diplomové práce je vyčerpávajícím způsobem popsaná RTL logika, která umožňuje popis systémů pracujících s reálným časem na nejnižší úrovni. Lze diskutovat i jiné typy temporálních logik, bohužel tyto nejsou v našem případě použitelné, jelikož neexistují žádné vhodné formální využitelné postupy pro aplikaci v grafickém popisu systémů.

Další kapitola popisuje dva nalezené způsoby hierarchického popisu systémů, které umožňují generovat popis modelu v logice RTL. Tento fakt je z hlediska zadání a zaměření práce klíčový. Existují samozřejmě i jiné přístupy, ty ale nenabízejí dostatečný formální základ nutný pro další použití v této práci.

Čtvrtá kapitola je velice směrodatná pro další vývoj. Bylo docíleno vhodného strukturálního návrhu grafického programu s ohledem na modulární strukturu. Tento model značně pomůže v následném objektovém návrhu systému.

Kapitola s pořadovým číslem pět detailně popsala způsob implementace nástroje a poukázala na všechny nabízené funkcionality. Dále je na několika obrázcích ukázán vzhled výsledného programu.

Poslední kapitola popisuje tři vybrané příklady systémů pracujících s reálným časem. Byla specifikována časová omezení systémů a také bezpečnostní požadavky, které musí být nutně splněny pro korektní běh. Nad těmito modely byly úspěšně vybudovány Modechart specifikace, nad kterými byl spuštěn verifikační proces.

Práce vychází ze studia značného množství dostupných podkladů a některé informace byly shledány mimo rámec dané publikace. Bylo dosaženo pozitivních výsledků ve směru výzkumu metod popisu systémů pracujících s reálným časem a jejich následné verifikace. Vytvořený nástroj pracoval bez problémů a jeho funkčnost byla ověřena na několika úlohách. Bylo docíleno generování RTL formulí popisujících Modechart model, které byly následně úspěšně verifikovány programem vyvíjeným Bc. Janem Fiedorem. Dále nad vzniklými formulemi byla provedena redukce jejich stavového prostoru. Byla vystavěna rozhodovací procedura nad Modechart modelem a byla shledána jako funkční.

Ikdyž je patrné, že práce pokrývá značnou část problematiky, naskýtají se i jisté možnosti dalšího rozšíření. Tato práce je základem navazujícího doktorského studia a proto bude nutno zvážit následující:

- Generování z Modechart modelu logických formulí, které bude následně možno ověřet oproti bezpečnostnímu tvrzení s použitím SMT solveru.
- Implementaci modulu umožňujícího simulaci běhu systému na základě analýzy SEP a CG grafu.
- Export grafů do formátu XML a následné zpracování dalšími programy např. pro visualizaci.

- Vystavění rozhodovacích procedur pro další třídy problémů analýzy Modechart specifikace.
- Vylepšení algoritmu hledání separací mezi uzly SEP grafu a použití ukládání mezivýsledků.
- Převod Modechart modelu do formy Kripkeho struktury a aplikace dalších verifikačních technik.

Literatura

[1] Aprvrille, L.; Courtiat, J.: TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit. *IEEE Transactions on Software Engineering*, ročník 30, č. 7, 2004: s. 473–487, dostupné na URL: <http://oatao.univ-toulouse.fr/282/1/De_Saqui_Sannes_282.pdf>.

[2] Aruchamy, G.; Cheng, A.: Translating Real-Time UML Timing Constraints into Real-Time Logic Formulas. Technická zpráva, University of Houston, TX, USA, Červen 2006, dostupné na URL: <http://www.cs.uh.edu/Preprints/2006/uhcs0607.pdf>.

[3] Cheng, A.: *Real-Time Systems: Scheduling, Analysis, and Verification*. John Willey, Inc., 2002, ISBN 0-471-18406-3.

[4] Jahanian, F.; Mok, A.: Formal Specification of Real-Time Systems. Technická zpráva, The University of Texas at Austin, TX, USA.

[5] Jahanian, F.; Mok, A.: Modechart: A specification Language for Real-Time Systems. *IEEE Transactions on Software Engineering*, ročník 20, č. 12, 1994: s. 933–947, dostupné na URL: <http://www2.computer.org/portal/web/csdl/doi/10.1109/32.368134>.

[6] JPF: Java Plugin Framework [ONLINE]. Last modified: 2007-07-02 21:33. [cit. 2009-04-25]. Dostupné na URL: <http://jpf.sourceforge.net/>.

[7] Puchol, C.; Stuart, D.: An Operational Semantics and a Compiler for Modechart Specification. Technická zpráva, The University of Texas at Austin, TX, USA, dostupné na URL: <http://www.cs.utexas.edu/ftp/pub/techreports/tr95-37.ps.gz>.

[8] Wikipedia: Model view controller [ONLINE]. Last modified on 22 May 2009. [cit. 2009-04-30]. Dostupné na URL: <http://en.wikipedia.org/wiki/Model-view-controller>.

[9] Yank, J.; ; Mok, A.; aj.: A New Generation Modechart Verifier. Technická zpráva, The University of Texas at Austin, TX, USA, dostupné na URL: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=516208>.